УДК 004.4

Титов Феликс Александрович,

магистрант, МИРЭА – Российский Технологический Университет, г. Москва

АНАЛИЗ И ОПТИМИЗАЦИЯ АРХИТЕКТУРНЫХ UI-КОМПОНЕНТОВ НА ОСНОВЕ ОБОБЩЕННЫХ ТИПОВ

Аннотация. В статье рассматриваются подходы к проектированию архитектуры библиотеки UI-компонентов для iOS-приложений на основе обобщенных типов. Описывается необходимость унификации элементов интерфейса, приводятся преимущества использования типизированных конфигураций компонентов, и демонстрируются способы повышения скорости разработки и поддержки продуктов на базе единой дизайн-системы.

Annotation. The article discusses approaches to designing the architecture of UI component library for iOS applications based on generalized types. It describes the necessity of unification of interface elements, gives the advantages of using typed component configurations, and demonstrates ways to increase the speed of product development and support on the basis of a unified design system.

Ключевые слова: UI, библиотека, iOS, SwiftUI, UIKit, View, Content, Container, Фреймворк.

Key words: UI, library, iOS, SwiftUI, UIKit, View, Content, Container, Framework.

Введение

В современном мире цифровых продуктов и приложений пользователи предъявляют высокие требования к удобству интерфейса, стабильности его работы и скорости реализации новых функций. Особенно остро эта задача стоит перед корпоративными приложениями, где интерфейс должен не только удовлетворять строгим стандартам компании, но и гибко адаптироваться к постоянно изменяющимся бизнес-требованиям. В таких условиях разработка единой, масштабируемой и переиспользуемой дизайн-системы становится критически важным фактором успеха программного продукта.

Постановка проблемы

Одной из современных и актуальных тенденций проектирования пользовательских интерфейсов является компонентный подход. При его использовании приложение разбивается на повторно используемые элементы (компоненты), каждый из которых обладает собственными конфигурациями и состояниями. Такой подход значительно сокращает время разработки, улучшает качество кода и позволяет соблюсти визуальную и функциональную согласованность интерфейсов.

Однако, несмотря на очевидные преимущества компонентного подхода, его практическое применение в мобильных приложениях для платформы iOS сталкивается с определёнными трудностями. В частности, возникает сложность поддержки единой дизайн-системы при одновременном использовании разных UI-фреймворков (например, UIKit [1] и SwiftUI [2]). Традиционные решения зачастую не обеспечивают достаточную гибкость настройки, требуют больших трудозатрат на поддержку и слабо адаптируются к изменениям.

В связи с этим актуальной является задача проектирования архитектуры библиотеки компонентов, которая обеспечивала бы гибкость конфигурирования, высокую степень переиспользования кода и простоту интеграции с различными подходами к разработке интерфейсов на платформе iOS. В настоящей статье предлагается подход, основанный на применении обобщенных типов (дженериков), позволяющий решать обозначенные проблемы, тем самым оптимизируя процесс разработки пользовательских интерфейсов.

Обзор существующих решений и формулировка задачи исследования

В настоящее время разработка пользовательских интерфейсов для платформы iOS осуществляется с использованием различных подходов и инструментов. Основными из них являются UIKit и SwiftUI, предоставляемые компанией Apple. UIKit, являясь более зрелым фреймворком, предоставляет широкие возможности для создания интерфейсов, однако требует значительных усилий для обеспечения переиспользуемости компонентов и поддержки различных состояний. SwiftUI, в свою очередь, предлагает декларативный

подход к разработке интерфейсов, упрощая процесс создания и сопровождения UI-компонентов. Однако, несмотря на преимущества, SwiftUI всё ещё развивается и не всегда обеспечивает необходимую гибкость и совместимость с существующими проектами на UIKit.

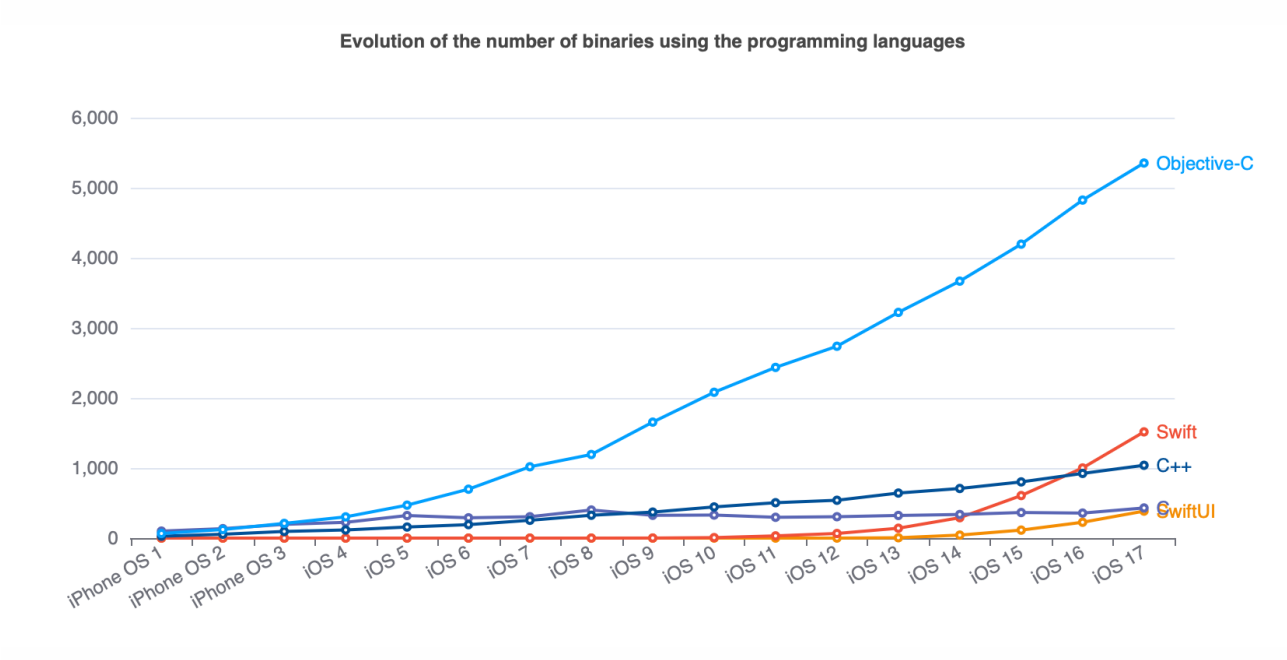

Рисунок 1 – Статистика использования фреймворков SwiftUI и UIKit (Swift + Objective-C) [3]

Существуют также сторонние библиотеки компонентов, такие как Carbon [4], Ring UI [5] и Material Components for iOS [6]. Библиотека Carbon предоставляет декларативный подход к созданию компонентов для UITableView и UICollectionView, вдохновлённый SwiftUI и React. Она облегчает создание и поддержку сложных интерфейсов, однако ориентирована преимущественно на списковые представления и не охватывает весь спектр UI-компонентов. Ring UI, разработанная компанией JetBrains, представляет собой набор компонентов для веб-приложений, но её архитектурные принципы могут быть адаптированы для мобильной разработки. Material Components for iOS реализует принципы Material Design от Google, однако с 2021 года находится в режиме поддержки и не получает активных обновлений.

Таблица 1 – Сравнение библиотек

Критерий	Carbon	Material UI	Ring UI

Поддержка iOS	На базе UIKit	Архив	Отсутствует
Открытый исходный код	Да	Да	Да
Спецификация UI-компонентов	Подробная	Архив	Частичная, с большим акцентом на Web
Соответствие Apple HIG	Да	Нет, ориентирована на Android UX	Нет, ориентирована на Desktop
Поддержка Accessibility	Активно поддерживается	Отсутствует для iOS	Реализация вручную

Анализ существующих решений показывает, что ни одно из них не предоставляет универсального подхода к созданию переиспользуемых и гибко настраиваемых UI-компонентов, полностью удовлетворяющего требованиям современных iOS-приложений. Это обуславливает необходимость разработки новой архитектуры библиотеки компонентов, основанной на использовании обобщённых типов (дженериков), обеспечивающей высокую степень переиспользуемости, гибкость конфигурации и простоту интеграции с различными фреймворками.

Описание предлагаемого архитектурного подхода

Предлагаемый архитектурный подход основывается на использовании обобщённых типов (дженериков) и протоколов, которые помогают создавать универсальные, масштабируемые и гибко настраиваемые компоненты пользовательского интерфейса для приложений на платформе iOS.

Архитектурное решение состоит из трех четко выделенных уровней:

Первый уровень (View) - отвечает за визуальное представление компонента и взаимодействие с пользователем. На этом уровне реализуются конкретные элементы интерфейса (например, кнопки, текстовые поля или

переключатели). Важно, что каждый элемент на этом уровне реализует общий протокол, который задает требования к структуре компонента и позволяет стандартным образом управлять его конфигурацией.

Второй уровень (Content) - представляет собой специальные структуры, которые описывают конфигурацию компонента, его внешний вид и возможные состояния (активное, неактивное, загружающее). Содержание этого уровня позволяет гибко настраивать интерфейсные элементы без необходимости изменения их внутренней логики. Таким образом, изменения внешнего вида или поведения компонентов осуществляются максимально просто и безопасно, не затрагивая базовую реализацию.

Третий уровень (Container) - выполняет роль связующего звена между уровнями View и Content. Именно здесь происходит передача конфигурации (состояния) во внутренний визуальный компонент и управление его жизненным циклом. Контейнер отвечает за то, чтобы внутренний компонент получал необходимые данные в корректном виде, а также за обработку событий взаимодействия с пользователем (например, нажатий, свайпов и других действий).

Использование обобщённых типов на уровне Container позволяет абстрагироваться от конкретного типа визуального компонента, что значительно повышает гибкость архитектуры. Вместо того чтобы создавать отдельные контейнеры для каждого типа визуального элемента, используется один универсальный контейнер, способный работать с любым элементом, соответствующим заданным протоколам и структурам конфигураций [7].

Далее будут описаны преимущества предлагаемого подхода.

1. Универсальность. Протоколы и дженерики в компонентах позволяют легко интегрироваться в различные проекты и могут быть использованы без существенных модфикаций;
2. Гибкость конфигурации. Возможность гибко изменять поведение компонентов, их внешний вид через мгновенное обновление

конфигурации компонента, что позволяет также не вмешиваться в код самого компонента;

3. Гибкое тестирование. Предложенный подход позволяет довольно просто протестировать компонент в различных его состояниях, меняя различные свойства его конфигурации.

Также необходимо учитывать, простоту масштабирования и адаптации интерфейсов к быстро меняющимся бизнес-требованиям.

Результат применения предложенного подхода

Для оценки эффективности предлагаемого архитектурного подхода, основанного на использовании обобщенных типов и протоколов, было проведено сравнение его применения с традиционным подходом, который предполагает индивидуальную разработку компонентов пользовательского интерфейса в каждом конкретном случае.

Одним из основных критериев оценки стала скорость разработки типичных интерфейсных элементов, таких как кнопки, текстовые поля, переключатели и индикаторы состояний. Для проведения эксперимента были выделены две команды разработчиков, сопоставимые по квалификации и опыту. Первая команда использовала традиционный подход (на основе UIKit с индивидуальной реализацией каждого компонента), а вторая команда применяла библиотеку компонентов, реализованную на основе предлагаемого архитектурного подхода [8].

Результаты эксперимента показали следующие преимущества предлагаемого подхода:

1. Уменьшение времени разработки и интеграции

Благодаря заранее определённым конфигурациям компонентов и их стандартизированным протоколам, время на разработку новых интерфейсных элементов существенно сократилось. В среднем, для реализации типичных экранов и компонентов пользовательского интерфейса, таких как формы авторизации, профили пользователей и экраны настроек, время сокращалось на

40–50% по сравнению с традиционным подходом. Это стало возможным за счёт переиспользования уже имеющихся элементов и упрощения конфигурирования.

2. Сокращение количества повторяющегося кода

Важным преимуществом предложенного подхода стало значительное уменьшение дублирования кода. В ходе анализа выяснилось, что объём кода сократился примерно на 30–35%, благодаря универсальности компонентов, которые стали легко настраиваться через структуры конфигураций. Это также положительно повлияло на чистоту кода, сделав его более понятным и читаемым для новых разработчиков, подключающихся к проекту.

3. Упрощение поддержки и развития проекта

Введение универсального подхода на основе дженериков позволило упростить поддержку уже реализованных компонентов. При возникновении необходимости внести изменения во внешний вид или поведение какого-либо элемента, разработчикам больше не нужно было переписывать реализацию каждого компонента отдельно. Вместо этого достаточно было обновить общую конфигурацию, применяемую ко всем элементам, реализующим конкретный протокол. Это сократило количество ошибок и снизило риски возникновения новых багов в проекте.

4. Удобство интеграции с разными подходами разработки интерфейсов

Компоненты, реализованные с применением дженериков и протоколов, легко интегрировались в уже существующие проекты, минимизируя трудозатраты и обеспечивая возможность использования единой дизайн-системы в разных модулях.

Заключение

В результате проведенного исследования был предложен и подробно рассмотрен архитектурный подход к проектированию библиотеки компонентов пользовательского интерфейса для мобильных приложений на платформе iOS, основанный на использовании обобщенных типов (дженериков) и протоколов. Данный подход направлен на решение существующих проблем компонентного подхода, связанных с ограниченной гибкостью, избыточностью кода,

сложностью поддержки и адаптации дизайн-систем в условиях корпоративной разработки.

В ходе работы были проанализированы существующие на рынке решения для построения UI-библиотек, такие как Carbon Components for iOS, Ring UI от JetBrains и Material UI. Анализ показал, что большинство решений предлагают ограниченные возможности адаптации и зачастую либо ориентированы на узкий круг задач, либо не поддерживают платформу iOS в полной мере.

Описанный подход включает использование универсальной структуры компонента, способного конфигурироваться через заранее заданные конфигурации и протоколы. Это позволило существенно сократить время разработки типовых интерфейсных элементов (в среднем на 40–50%), уменьшить объем повторяющегося кода (на 30–35%), упростить поддержку и дальнейшее развитие проекта.

Практическое применение изложенной архитектуры позволило повысить эффективность работы команды разработчиков и добиться большей согласованности интерфейсов, улучшить их удобство и привлекательность для конечных пользователей.

Кроме того, методология продемонстрировала свою стабильность и гибкость в случае динамично меняющихся бизнес-требований.

Литература

1. Apple Developer Documentation. UIKit. – URL: https://developer.apple.com/documentation/uikit (дата обращения: 29.04.2024).

2. Apple Developer Documentation. SwiftUI. – URL: https://developer.apple.com/documentation/swiftui (дата обращения: 29.04.2024).

3. Tibor Bödecs. State of Swift and SwiftUI in iOS 17 – 2023. – URL: https://blog.timac.org/2023/1019-state-of-swift-and-swiftui-ios17/ (дата обращения: 30.04.2024).

4. IBM Carbon Design System. Carbon Components for iOS. – URL: https://carbondesignsystem.com/developing/ios-tutorial/overview (дата обращения: 29.04.2024).

5. JetBrains Ring UI: JetBrains Web UI components. – URL: https://jetbrains.github.io/ring-ui/ (дата обращения: 29.04.2024).

6. Material UI: React Components Library. – URL: https://mui.com (дата обращения: 29.04.2024).

7. J. Caballero, N. Moreno, Software architectures for design systems and component libraries: A systematic review, Journal of Systems and Software, vol. 199, 2023, Article 111650.

8. A. Yaqoob, R. Zafar, M. Abbas, et al., Design systems and their impact on user interface development in large enterprises, International Journal of Human-Computer Interaction, 2022, vol. 38, issue 6, pp. 543-557.