Программные системы и вычислительные методы
Правильная ссылка на статью:
Петровский А.А., Рысин М.Л. Реализация поведения Drag&Drop в Android-приложении на основе API обработки жестов // Программные системы и вычислительные методы. 2025. № 2. DOI: 10.7256/2454-0714.2025.2.74122 EDN: FIAYDF URL: https://nbpublish.com/library_read_article.php?id=74122

Реализация поведения Drag&Drop в Android-приложении на основе API обработки жестов

Петровский Александр Андреевич

бакалавр; Институт информационных технологий; МИРЭА-Российский технологический университет

119454, Россия, г. Москва, пр-кт Вернадского, д. 78

petrovskiy.a.a@edu.mirea.ru

Рысин Михаил Леонидович

кандидат педагогических наук

доцент; кафедра математического обеспечения и стандартизации информационных технологий; МИРЭА-Российский технологический университет

119454, Россия, г. Москва, пр-кт Вернадского, д. 78

rysin@mirea.ru

Статья из рубрики "Методы, языки и модели человеко-машинного взаимодействия"

DOI:

10.7256/2454-0714.2025.2.74122

EDN:

FIAYDF

Дата направления статьи в редакцию:

16-04-2025

Аннотация: Объектом исследования выступает организация перемещения объектов пользовательского интерфейса (User Interface, UI) в Android-приложениях. Предметом исследования является разработка программного решения для реализации поведения Drag&Drop в мобильных Android-приложениях с использованием современного фреймворка пользовательского интерфейса Jetpack Compose. Целью представленной работы является создание гибкого и интуитивно понятного механизма взаимодействия пользователя с интерфейсом мобильных Android-приложений. Основные результаты

исследования включают: 1. Разработку набора Composable функций для управления Drag&Drop состоянием объектов пользовательского интерфейса. 2. Объединение поведения «источника» и «получателя» перемещаемых UI-объектов с возможностью декорирования. 3. Преодоление ограничений штатных инструментов фреймворка Jetpack Compose. 4. Создание механизма обработки пользовательских жестов перемещения. 5. Формирование универсального подхода к реализации интерактивного взаимодействия с элементами интерфейса. Методология основана на применении архитектурного паттерна MVI (Model-View-Intent), который обеспечивает эффективное управление состоянием интерфейса, и использовании объектно-ориентированных паттернов проектирования, в частности, паттерна "декоратор". Методы исследования включают анализ существующих подходов к реализации Drag&Drop, проектирование программного решения, разработку прототипа и его апробацию в рамках мобильного приложения. Научная новизна исследования заключается в разработке инновационного подхода к организации Drag&Drop взаимодействия, который позволяет преодолеть ограничения штатных инструментов фреймворка Jetpack Compose. Предложенное авторами решение характеризуется: - полной изолированностью компонентов Drag&Drop; - возможностью декорирования перемещаемых UI-объектов; - гибкой настройкой поведения источника и приемника объектов интерфейса; - отсутствием жестких связей между компонентами пользовательского интерфейса. Практическая значимость работы заключается в разработке инструментария, который может быть успешно применен в различных мобильных программных проектах, требующих реализации сложных пользовательских взаимодействий. Выводы исследования демонстрируют эффективность предложенного решения в преодолении существующих ограничений Jetpack Compose и открывают новые возможности для создания более динамичных и удобных пользовательских интерфейсов в мобильных приложениях.

Ключевые слова:

мобильное приложение, Android, технология Drag-and-Drop, пользовательский интерфейс, обработка жестов, Jetpack Compose, паттерн MVI, паттерн декоратор, модель-представление-намерение, Composable функция

Введение

Поведение Drag&Drop представляет собой инструмент взаимодействия пользователя с компьютерной программой, в результате которого происходит перемещение видимых объектов по экрану. Такое поведение позволяет создать отзывчивый и удобный пользовательский интерфейс (User Interface, UI), т.к. имеет аналогию с манипуляцией объектами в реальном мире [1].

Рассмотрим задачу применения поведения Drag&Drop к любым элементам пользовательского интерфейса (UI-элементам или UI-объектам). Решением станет набор декорирующих Composable функций для управения единым состоянием процесса перемещения.

В основу решения положим паттерн MVI (Model-View-Intent); используем для работы с пользовательским интерфейсом фреймворк Jetpack Compose. Выбор указанных инструментов обусловлен тем, что на момент написания статьи Jetpack Compose является целевым решением от компании Google [2], а паттерн MVI более всего подходит для работы с Jetpack Compose, т.к. реализует работу с единым состоянием экрана при

помощи получения событий Intent от слоя отображения [3].

В настоящее время для реализации поведения Drag&Drop согласно официальной документации рекомендовано использование двух модификаторов – dragAndDropSource и dragAndDropTarget, которые соответственно позволяют применять к Composable поведение либо источника объектов (элементов, которые могут быть перемещены пользователем), либо поведение слота (элемента интерфейса, считывающего события завершения процесса перетаскивания объекта) [4].

Основная проблема этого решения заключается в конфликте этих двух модификаторов при применении к одному и тому же Composable объекту, а также к объектам, находящимся в одной иерархии вложенности. Суть проблемы заключается в том, что единовременно, к объекту (иерархии объектов) может быть применено лишь одно из двух свойств – «перетаскиваемый» объект или позиция для получения «перетаскиваемого» объекта. Таким образом, при попытке переместить объект в область экрана, с которой до этого был взят объект происходит конфликт – позиция, считывающая перемещения, перестает реагировать на события Drag&Drop. Частично указанная проблема зафиксирована самими разработчиками [5].

Таким образом целевое решение для организации поведения Drag&Drop позволяет реализовать лишь однонаправленное перемещение объектов, при этом не давая возможности объединить поведение источника и получателя объектов перемещения, обеспечив полную обработку событий.

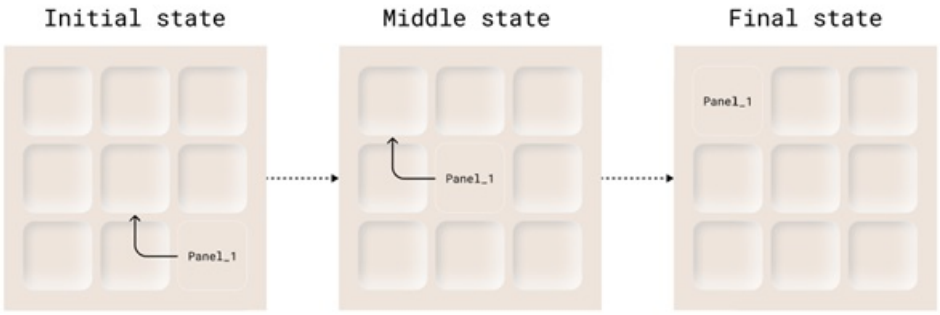

Рисунок 1 – Визуализация процесса перемещения объектов интерфейса

Такое поведение является интуитивным для пользователя, так как позволяет забрать объект из той позиции на экране, в которую он был перемещен ранее. В качестве конкретного примера, приводится проект, в рамках которого будет рассмотрен собственный подход к организации Drag&Drop. В проекте необходимо реализовать матрицу из позиций, между которыми пользователь должен иметь возможность беспрепятственно перемещать панели, в т.ч. иметь возможность «положить» и «взять» панель с одной и той же позиции.

На рис. 1 приведен сценарий перемещения элементов интерфейса, который должен быть реализуем при помощи описываемого программного решения.

Определение требований к программному решению

В качестве ключевых требований к инструменту, реализующему поведение Drag&Drop определим следующие:

· Изолированность. Решение должно представлять из себя изолированный инструмент – класс или набор методов, позволяющий интегрировать поведение Drag&Drop в любой

экран проекта без жестких связей как с элементами слоя отображения, так и слоев Domain и Data [6, с. 202-209]. Инструмент должен обеспечивать лишь поведение и не ограничивать характеристики объектов, участвующих в процессе Drag&Drop.

· Функциональность. Решение считаем функционально полным, если будет предоставляться возможность для настройки визуальной составляющей перемещаемого объекта, а также наделять Composable объекты свойствами источника и приемника Drag&Drop объектов.

· Инкапсуляция. Инструмент не должен предоставлять во вне особенности реализации своего поведения. А именно, все процессы, относящиеся к работе с характеристиками перемещаемого объекта, должны быть сокрыты от тех мест, в которых будет использоваться инструмент.

Описав требования к рассматриваемому инструменту (Drag&Drop Utils), определим ключевые компоненты инструмента (рис. 2):

· Сущность, хранящая информацию о сеансе Drag&Drop. Данная сущность должна содержать в себе всю необходимую информацию для обеспечения перемещения объекта.

· Composable функция источника. Должна обрабатывать все события процесса Drag&Drop и уведомлять внешних подписчиков об этих событиях.

· Composable функция слота. Должна формировать матрицу возможных позиций для приема объекта Drag&Drop.

· Composable функция подписки на общее состояние и предоставляющая интерфейс для декорирования объекта Drag&Drop.

Таким образом инструмент будет состоять из трех основных функций, реализующих паттерн декоратор [7, с. 209-220]. Данные функции будут присваивать получаемому содержимому поведение Drag&Drop.

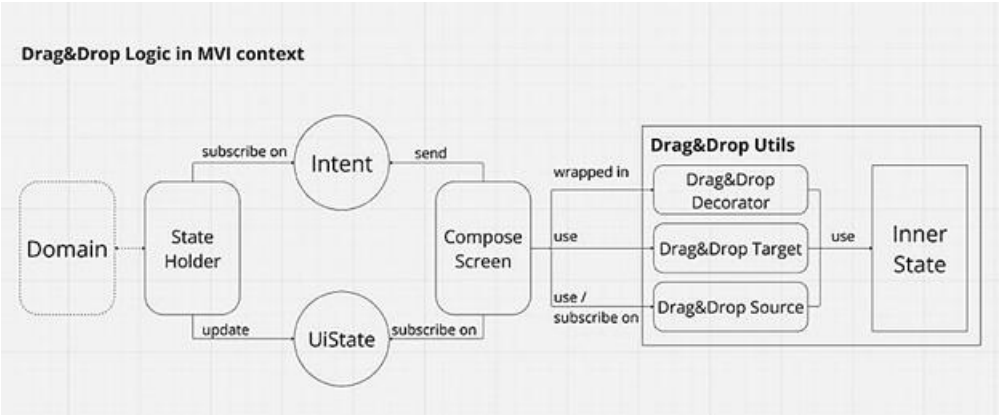

Рисунок 2 – Диаграмма компонентов в контексте MVI

Стоит также отметить, что благодаря использованию паттерна MVI обработку событий перемещения, которые относятся к бизнес-логике приложения, можно полностью делегировать в обработчик состояния благодаря отправке событий пользовательского интерфейса Intent.

На рис. 2 отображены следующие компоненты:

· ComposeScreen – экран с пользовательским интерфейсом. Представляет собой слой отображения, на котором организуется поведение Drag&Drop.

· Intent – событие пользовательского интерфейса, которое обрабатывается в StateHolder.

· StateHolder – обработчик пользовательских событий, отвечающий за изменение состояния интерфейса.

· UiState – состояние пользовательского интерфейса, на обновления которого подписан экран.

· Drag&Drop Utils – описываемый в статье инструмент, предоставляющий три декоративные функции и хранящий приватное состояние InnerState.

· Domain – абстрактное определение доменного слоя проекта, детали которого не влияют на работу рассматриваемого инструмента.

Практические аспекты реализации

Необходимо отметить, что для реализации перемещения необходимо создать общее состояние, которое будет доступно для всех composable функций внутри экрана. Для реализации данного поведения предлагается использовать CompositionLocalProvider[8], который будет размещен в корневой функции-декораторе. При этом экземпляр класса состояния сделаем приватным в файле, реализующем поведение Drag&Drop. Благодаря такому решению область видимости состояния Drag&Drop будет ограничена только рамками функций, отвечающих за логику перемещения, что позволит нам выполнить требование к инкапсуляции.

Для выполнения требования изолированности предложим следующее решение: все функции, обеспечивающие поведения Drag&Drop будут принимать в себя последним аргументом лямбда-выражение с аннотацией @Composable, тем самым позволяя передавать внутрь любую composable функцию, т.е. присваивать поведение перемещения к любым элементам интерфейса без исключения.

Приведем программный код спроектированного решения с пояснениями к наиболее важным его частям.

В листинге 1 показан класс состояния процесса Drag&Drop и создание его экземпляра.

Листинг 1. Класс состояния, содержащий свойства перетаскиваемого объекта

```
private class DraggableItemInfo {

var dndObject by mutableStateOf<(@Composable (Int) -> Unit)?>(null)

var dndObjectSize by mutableIntStateOf(0)

var isDragging: Boolean by mutableStateOf(false)

var dndStartOffset by mutableStateOf(Offset.Zero)

var dndCurrentOffset by mutableStateOf(Offset.Zero)

var targetOffsetMap = mutableStateMapOf<Coordinate, Rect>()

}
```

```
private val LocalDraggableItemInfo =

compositionLocalOf { DraggableItemInfo() }
```

В классе последовательно объявлены: composable функция перемещаемого объекта, размеры объекта, состояние объекта – находится ли он в перемещении или нет, начальное смещение объекта по двум осям, текущее смещение объекта по двум осям, а также карта потенциальных позиций, на которые объект может быть перемещен.

Карта позиций представляет собой словарь, где значением является координатная область, в рамках которой считывается окончание процесса перемещения, а ключом – координата слота, которая возвращается в событии для обработки на уровне бизнес-логики.

В листинге 2 приведена корневая функция-декоратор, которая декларирует общее состояние, а также предоставляет интерфейс для декорирования перемещаемого объекта.

Листинг 2. Функция инициализации состояния и декорирования объекта

```
@Composable

internal fun DragAndDropStateHandler(

dndObjectDecorator: GraphicsLayerScope.() -> Unit,

content: @Composable BoxScope.() -> Unit,

) {

val state = remember { DraggableItemInfo() }

CompositionLocalProvider(LocalDraggableItemInfo provides state) {

Box(modifier = Modifier.fillMaxSize()) {

content()

if (state.isDragging) {

DraggableComposableDecoration(state, dndObjectDecorator)

}

}

}

}

@Composable

private fun DraggableComposableDecoration(

state: DraggableItemInfo,

dndObjectDecorator: GraphicsLayerScope.() -> Unit,

) {
```

```
var targetSize by remember { mutableStateOf(IntSize.Zero) }
Box(
modifier = Modifier
.graphicsLayer {
val offset = (state.dndStartOffset + state.dndCurrentOffset)
translationX = offset.x - (targetSize.width / 2)
translationY = offset.y - targetSize.height
dndObjectDecorator()
}
.onGloballyPositioned { targetSize = it.size }
) {
state.dndObject?.invoke(state.dndObjectSize)
}
}
```

В этой функции есть два ключевых момента. Первый – это инициализация локального состояния, которая происходит при помощи функции CompositionLocalProvider. Второй – непосредственно логика перемещения и декорирования, которая происходит в блоке graphicsLayer [9]. Здесь осуществляется считывание стартового и текущего смещения объекта и присвоение перемещаемому объекту данных свойств. Также вызывается лямбда-выражение dndObjectDecorator, в качестве объекта-получателя которому передается экземпляр graphicsLayer [10], тем самым применяя определенные вовне свойства к перемещаемому объекту. Фрагмент описания данных свойств, а также применения описываемой функции показан в листинге 3.

Листинг 3. Использование функции хранения состояния на примере экрана

```
@Composable
internal fun DisplayLevelScreen(
viewModel: DisplayLevelViewModel,
) {
val uiState = viewModel.uiState.collectAsState()
val intentHandler by rememberUpdatedState(viewModel::sendIntent)
DragAndDropStateHandler(
dndObjectDecorator = {
scaleX = DRAG_AND_DROP_SCALE_RATIO
scaleY = DRAG_AND_DROP_SCALE_RATIO
```

```
alpha = DRAG_AND_DROP_ALPHA
}
) {
DisplayLevelScreenContent(uiState, intentHandler)
}
}
```

Здесь можно явно увидеть, что контент всего экрана оборачивается в описанную выше функцию, а также реализуется декорирование объекта (изменяется размер и прозрачность). Важно, во-первых, отметить, что функция является лишь оберткой, не ограничивая функционал экрана, и, во-вторых, что декорирование перемещаемого объекта находится в зоне ответственности экрана, тем самым визуальные характеристики перемещаемого объекта могут гибко изменятся в зависимости от экрана.

Далее опишем функцию слота (листинг 4).

Листинг 4. Composable функция слота, формирующая карту позиций Drag&Drop

```
@Composable
internal fun DragAndDropTarget(
modifier: Modifier = Modifier,
coordinate: Coordinate,
content: @Composable (() -> Unit),
) {
val currentState = LocalDraggableItemInfo.current
Box(
modifier = modifier
.onGloballyPositioned { layoutCoordinates ->
currentState.dndObjectSize = layoutCoordinates.size.height
layoutCoordinates
.boundsInWindow()
.let { rect -> currentState.targetOffsetMap[coordinate] = rect }
},
contentAlignment = Alignment.Center
) {
content()
```

```
}

}
```

Можно заметить, что функция слота лаконично отвечает только за наполнение карты возможных целей перемещения объекта. На вход функция получает ключ позиции – Coordinate и, после этапа позиционирования объектов добавляет свою область в карту.

Пример использования будет представлен после описания функции источника объектов перемещения.

Наиболее важной является функция источника объектов перемещения, код которой представлен в листинге 5.

Листинг 5. Composable функция источника, считывающая пользовательские жесты

```
@Composable

internal fun DraggableView(

modifier: Modifier = Modifier,

onDragStart: () -> Unit = {},

onDragEnd: (Coordinate?) -> Unit = {},

onDragCancel: () -> Unit = {},

onClick: () -> Unit = {},

content: @Composable ((Int) -> Unit),

) {

var currentPosition by remember { mutableStateOf(Offset.Zero) }

val currentState = LocalDraggableItemInfo.current

Box(modifier = modifier

.onGloballyPositioned { layoutCoordinates ->

currentPosition = layoutCoordinates.localToWindow(Offset.Zero)

}

.pointerInput(Unit) {

detectDragGesturesAfterLongPress(

onDragStart = { startOffset ->

onDragStart.invoke()

currentState.isDragging = true

currentState.dndStartOffset = currentPosition + startOffset

currentState.dndObject = content
```

```
},
onDrag = { _, dragAmount ->
currentState.dndCurrentOffset += Offset(dragAmount.x, dragAmount.y)
},
onDragEnd = {
val offset = currentState.dndStartOffset + currentState.dndCurrentOffset
val targetOffsetMap = currentState.targetOffsetMap.toMap()
val targetCoordinate = extractCoordinateByOffset(targetOffsetMap, offset)
onDragEnd.invoke(targetCoordinate)
currentState.isDragging = false
currentState.dndCurrentOffset = Offset.Zero
},
onDragCancel = {
onDragCancel.invoke()
currentState.dndCurrentOffset = Offset.Zero
currentState.isDragging = false
}
)
}
.pointerInput(Unit) {
detectTapGestures {
onClick.invoke()
}
}
) {
content(currentState.dndObjectSize)
}
}
```

В первую очередь стоит отметить сигнатуру функции, в рамках которой передается три лямбда-выражения, которые необходимы для получения события перемещения и передачи его на уровень обработчика StateHandler.

При помощи модификатора считывания пользовательских жестов pointerInput [11] реализована обработка следующих событий: onDragStart, onDrag, onDragEnd и onDragCancel. Отдельно выведено считывание события простого нажатия – detectTapGestures.

Опишем каждое событие:

· onDragStart – событие долгого нажатия на объект. В рамках события передается сообщение о событии подписчику, состояние объекта isDragging (нахождение в процессе Drag&Drop) переводится в true, устанавливается начальное смещение и присваивается перемещаемый объект.

· onDrag – состояние перемещения объекта. В рамках события обновляется текущее смещение.

· onDragEnd – событие успешного завершения перемещения. Во-первых, в рамках события происходит сверка с картой целей и вычисляется попал ли объект в цель из сформированной карты. Далее эта информация передается подписчику в лямбде. После этого состояние объекта isDragging возвращается в состояние false, а также обнуляется текущее смещение.

· onDragCancel – событие отмены текущего перемещения другим жестом. В рамках этого события также обнуляется состояние объекта.

Таким образом получаем функцию для обработки события процесса Drag&Drop, которая обеспечивает само перемещение объекта за счёт управления локальным состоянием, а также уведомляет подписчика о всех необходимых событиях. При этом все лямбда-функции имеют реализацию по умолчанию, а значит подписчик может подписаться только на те события, которые ему необходимы.

В листинге 6 покажем composable функцию, представляющую собой комбинированный источник и слот для Drag&Drop.

Листинг 6. Использование функция источника и слота Drag&Drop на примере экрана

```
DragAndDropTarget(

modifier = Modifier.fillMaxSize(),

coordinate = Coordinate(x, y),

) {

if (containsPanel) {

DraggableView(

modifier = Modifier.fillMaxSize(),

onDragStart = { onDragStart(currentPanel, intentHandler) },

onDragEnd = { targetCoordinate -> onDragEnd(targetCoordinate, intentHandler) },

onClick = { intentHandler(DisplayLevelIntent.Click(currentPanel)) }

) { panelSize ->
```

```
DisplayLevelPanel(panelSize, currentPanel)

}

}

}
```

Как видно из листинга, наделение объекта поведением слота и источника Drag&Drop происходит лишь за счет создания иерархии вложенности функций без жестких связей. DragAndDropTarget задает ключ для карты, а DraggableView подписывается на события и передает их на уровень обработчика при помощи intentHandler. Для чистоты кода события onDragStart и onDragEnd вынесены в отдельные функции, которые показаны в листинге 7.

Листинг 7. Методы обработки событий процесса Drag&Drop

```
private fun onDragStart(panel: PanelUiModel, intentHandler: (DisplayLevelIntent) -> Unit) {

intentHandler(DisplayLevelIntent.DragAndDrop(Up(panel)))

}

private fun onDragEnd(

targetCoordinate: Coordinate?,

intentHandler: (DisplayLevelIntent) -> Unit,

) {

if (targetCoordinate == null) {

intentHandler(DisplayLevelIntent.DragAndDrop(Fail))

} else {

intentHandler(DisplayLevelIntent.DragAndDrop(Down(targetCoordinate)))

}

}
```

Как видно, события перемещения объектов передаются в формате Intent в StateHandler, где далее обрабатываются в соответствии с бизнес-логикой.

Описанное программное решение позволило на практике реализовать перемещение объектов по матрице позиций в мобильном приложении. Процесс перемещения поэтапно представлен на рис. 3.

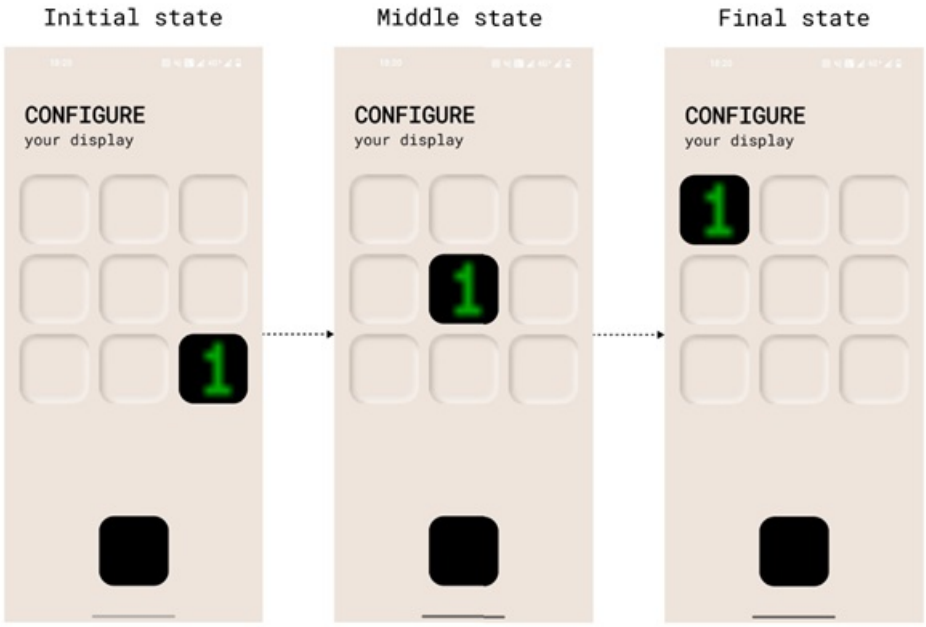

Рисунок 3 – Демонстрация применения программного решения

Выводы

Основные полученные результаты:

1. Разработан набор Composable функций для управления Drag&Drop состоянием объектов пользовательского интерфейса.

2. Объединены поведение «источника» и «получателя» перемещаемых UI-объектов с возможностью декорирования.

3. Преодолены ограничения штатных инструментов фреймворка Jetpack Compose.

4. Создан механизм обработки пользовательских жестов перемещения.

5. Сформирован универсальный подход к реализации интерактивного взаимодействия с элементами интерфейса.

Предложенное решение характеризуется:

- полной изолированностью компонентов Drag&Drop;

- возможностью декорирования перемещаемых UI-объектов;

- гибкой настройкой поведения источника и приемника объектов интерфейса;

- отсутствием жестких связей между компонентами пользовательского интерфейса.

Разработан инструментарий, который может быть успешно применен в различных мобильных программных проектах, требующих реализации сложных пользовательских взаимодействий.

Контрольные примеры работы демонстрируют эффективность предложенного решения в преодолении существующих ограничений фреймворка Jetpack Compose. Полученные результаты открывают новые возможности для создания более динамичных и удобных пользовательских интерфейсов в Android-приложениях.

Библиография

1. What is Drag and Drop? Drag and Drop explained [Электронный ресурс]. URL: https://goodspeed.studio/glossary/what-is-drag-and-drop-drag-and-drop-explained (дата обращения: 14.03.2025).
2. Jetpack Compose [Электронный ресурс]. URL: https://developer.android.com/compose (дата обращения: 18.03.2025).
3. Почему так удобно использовать паттерн MVI в KMM [Электронный ресурс]. URL: https://habr.com/ru/companies/kts/articles/729832/ (дата обращения: 18.03.2025).
4. Jetpack Compose: Drag and Drop [Электронный ресурс]. URL: https://developer.android.com/develop/ui/compose/touch-input/user-interactions/drag-and-drop (дата обращения: 20.03.2025).
5. Google IssueTracker: DragAndDropTarget [Электронный ресурс]. URL: https://issuetracker.google.com/issues/324280271 (дата обращения: 20.03.2025).
6. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения. – СПб.: Питер, 2024. – 352 с.
7. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Паттерны объектно-ориентированного программирования. – СПб.: Питер, 2022. – 448 с.
8. Jetpack Compose: CompositionLocal [Электронный ресурс]. URL: https://developer.android.com/develop/ui/compose/compositionlocal (дата обращения: 05.03.2025).
9. Jetpack Compose: GraphicsLayer [Электронный ресурс]. URL: https://developer.android.com/develop/ui/compose/graphics/draw/modifiers (дата обращения: 10.03.2025).
10. Kotlin: Scope-functions [Электронный ресурс]. URL: https://kotlinlang.org/docs/lambdas.html#invoking-a-function-type-instance (дата обращения: 11.03.2025).
11. Jetpack Compose: Pointer Input [Электронный ресурс]. URL: https://developer.android.com/develop/ui/compose/touch-input/pointer-input (дата обращения: 15.03.2025).

Результаты процедуры рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.
Со списком рецензентов издательства можно ознакомиться здесь.

Рецензируемое исследование посвящено применению технологии управления элементами программного обеспечения планшетов и смартфонов с операционной системой Android при помощи захвата, перетаскивания и бросания элементов в другое место с применением алгоритмов распознавания жестов.
Методология разработки базируется на применении архитектурного паттерна MVI (Model-View-Intent), который обеспечивает разделение обязанностей между моделью данных, компонентами пользовательского интерфейса и взаимодействием пользователя, а также фреймворка Jetpack Compose, упрощающего и ускоряющего создание мобильных приложений.
Актуальность работы обусловлена необходимостью развития инструментов взаимодействия пользователей с программами на мобильных устройствах для создания удобного интерфейса, имеющего аналогию с манипуляцией объектами в реальном мире.
Научная новизна работы – разработан инструментарий для применения в различных мобильных программных проектах, требующих реализации сложных пользовательских взаимодействий для создания более динамичных и удобных интерфейсов в Android-приложениях.

Структурно в тексте публикации выделены следующие разделы: Введение, Определение требований к программному решению, Практические аспекты реализации, Выводы и Библиография.

В публикации приведена визуализация процесса перемещения объектов интерфейса; сформулированы ключевые требования к инструменту, реализующему поведение Drag&Drop (изолированность, функциональность, инкапсуляция); определены ключевые компоненты инструмента (сущность, хранящая информацию о сеансе Drag&Drop, Composable функции источника, слота и подписки на общее состояние). В самом значительном по объему разделе «Практические аспекты реализации» приведены 7 фрагментов листинга программного кода спроектированного решения с пояснениями к наиболее важным его частям.

Библиографический список включает 11 источников, представленных преимущественно интернет-ресурсами на английском и русском языках, а также двумя многостраничными публикациями по рассматриваемой теме. В тексте публикации имеются адресные отсылки к списку литературы, подтверждающие наличие апелляции к оппонентам.

Из замечаний стоит отметить следующие. Во-первых, название работы, по мнению рецензента, лучше сформулировать без использования сложносокращенных слов на английском языке – ведь статья подготовлена на русском языке, да и тема, отражающая её содержание, должна быть понятна широкому кругу читателей, а не только узкопрофильным специалистам. Во-вторых, в статье не сформулированы цель и задачи исследования, его предмет и объект, не отражены рабочая гипотеза и методы проведения исследования, не акцентировано внимание на элементах приращения научного знания. В-третьих, авторами не выполнены рекомендации по оформлению списка литературы, принятые издательством: «Рекомендованный объем списка литературы … – не менее 20 источников, который должен содержать: не менее трети зарубежных источников; не менее половины работ, изданных в последние 3 года. В списке литературы не указываются … Интернет-источники, включая информацию с сайтов, а также статьи на сайтах и в блогах»). В-четвертых, фрагменты листинга программного кода представляется нужным оформить в виде рисунков с соответствующей нумерацией и названиями.

Тема статьи актуальна, соответствует тематике журнала «Программные системы и вычислительные методы», но требует доработки в соответствии с высказанными замечаниями.

Результаты процедуры повторного рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.

Со списком рецензентов издательства можно ознакомиться здесь.

Рецензируемая статья посвящена проблеме разработки интерфейсов приложений для мобильных устройств с помощью известных инструментальных решений, имеющих функциональные ограничения. Авторы подчеркивают, что одним из приоритетов является удобство пользователя при выполнении наиболее распространенного сценария, формулируют требования к программному решению.

Структура статьи требует доработки для приведения в соответствии с требованиями к научным публикациям.

Экспериментальная часть, описание исходных данных и количественные оценки результатов отсутствуют.

Стиль изложения близок к публицестическому. Имеются иллюстрации, качество достаточное.

Библиография содержит 11 источников, преимущественно на интернет-ресурсы. Отсутствуют публикации в рецензируемых журналах. Ссылки по тексту имеются.
Замечания.
Сформулировать цели и задачи исследования, новизну полученных результатов.
В структуре статьи четко выделить основные разделы – обзор аналогов, материалы и методы, обсуждение результатов.
При обзоре аналогичных продуктов необходимо использовать публикации в отечественных и зарубежных научных журналах, отмечая их ограничения и тем самым обосновывая актуальность выполняемого Авторами исследования.
Рис. 1 необходимо переделать с учетом языка статьи.
Перечисление требований сформулировано формально, не ясна роль в контексте выполняемого авторами исследования.
Необходимо использовать принятую терминологию, не смешивая формулировки на разных языках.
Рис. 2 вызывает сомнение и с учетом формы представления выглядит заимствованным, однако ссылки отсутствуют.
Описание разработки Авторов не имеет четкой структуры, все приведенные обоснования должны быть приведены в логической последовательности.
Приведенные листинги (в представленной форме) кажутся избыточными для научной статьи. Рекомендуется насколько это возможно заменить их на обоснование выбора используемых инструментов или последовательности действий, завершив анализом полученных результатов. Такая форма позволит оценить результативность используемых инструментов и одновременно вклад авторов. Возможно использование блок-схем.
Необходимо четко сформулировать задачу исследования авторов и результат. Каким образом (инструменты, статистика, критерии) оценивалась работоспособность предложенных решений, возможно в сравнении с другими известными средствами. Если оценка выполнялась пользователями, необходимо привести количественные результаты этой оценки. Если результат оценивали эксперты, необходимо также указать критерии.
Авторы отмечают использование элементов визуального оформления (различная прозрачность и размер), в этом случае оценка результатов может выполняться на небольшой выборке с помощью A/B тестирования.
Рис. 3 – представленное программное решение предназначено для приложения какого рода? (учебный продукт, коммерческое задание, разработка блока в рамках проекта и пр).
В заключительной части статьи необходимо сформулировать результаты с позиции развития метода и решения конкретной задачи, сформулированной авторами.
Библиографию необходимо дополнить ссылками на статьи в рецензируемых журналах (отечественных, зарубежных).
Статья может быть опубликована после внесения правок и повторного рецензирования.

Результаты процедуры окончательного рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.
Со списком рецензентов издательства можно ознакомиться здесь.

Представленная статья на тему «Реализация поведения Drag&Drop в Android-приложении на основе API обработки жестов» соответствует тематике журнала «Программные системы и вычислительные методы» и посвящена вопросу применения поведения Drag&Drop к любым элементам пользовательского интерфейса (UI-элементам или UI-объектам). В качестве решения авторами рассмотрен паттерн MVI (Model-View-

Intent), используемый для работы с пользовательским интерфейсом фреймворк Jetpack Compose. Выбор указанных инструментов обусловлен тем, что на момент написания статьи Jetpack Compose является целевым решением от компании Google, а паттерн MVI более всего подходит для работы с Jetpack Compose, т.к. реализует работу с единым состоянием экрана при помощи получения событий Intent от слоя отображения. В статье авторами визуализация процесса перемещения объектов интерфейса представлена в виде рисунка.

Авторами в качестве ключевых требований к инструменту, реализующему поведение Drag&Drop определены следующие:

· Изолированность.

· Функциональность.

· Инкапсуляция.

Также авторами определены ключевые компоненты инструмента:

· Сущность, хранящая информацию о сеансе Drag&Drop.

· Composable функция источника.

· Composable функция слота.

· Composable функция подписки на общее состояние и предоставляющая интерфейс для декорирования объекта Drag&Drop.

Для выполнения требования изолированности авторы предлагают следующее решение: все функции, обеспечивающие поведения Drag&Drop будут принимать в себя последним аргументом лямбда-выражение с аннотацией @Composable, тем самым позволяя передавать внутрь любую composable функцию, т.е. присваивать поведение перемещения к любым элементам интерфейса без исключения.

В качестве результатов исследования авторами указаны:

1. Разработка набора Composable функций для управления Drag&Drop состоянием объектов пользовательского интерфейса.

2. Объединение поведения «источника» и «получателя» перемещаемых UI-объектов с возможностью декорирования.

3. Преодоление ограничений штатных инструментов фреймворка Jetpack Compose.

4. Создание механизма обработки пользовательских жестов перемещения.

5. Формирование универсального подхода к реализации интерактивного взаимодействия с элементами интерфейса.

Список литературы представлен российскими и зарубежными источниками по теме исследования. Стиль и язык изложения материала является достаточно доступным для широкого круга читателей. Статья по объему соответствует рекомендуемому объему от 12 000 знаков.

Статья достаточно структурирована - в наличии введение, выводы, внутреннее членение основной части (определение требований к программному решению, практические аспекты реализации).

К недостаткам можно отнести следующие моменты: из содержания статьи не прослеживается научная новизна, отсутствует предмет исследования, цель исследования.

Рекомендуется четко обозначить научную новизну исследования, сформулировать предмет и цель исследования. Также будет целесообразным добавить о перспективах дальнейшего исследования и обосновать практическую значимость.

Статья «Реализация поведения Drag&Drop в Android-приложении на основе API обработки жестов» требует доработки по указанным выше замечаниям. После внесения поправок рекомендуется к повторному рассмотрению редакцией рецензируемого научного журнала.