DOI: 10.15514/ISPRAS-2026-38(3)-48

Автоматизация портирования Android-приложений в экосистему HarmonyOS NEXT с использованием формальных спецификаций

И.А. Миленин, ORCID: 0009-0000-6151-4038<svakun@gmail.com>
В.М. Ицыксон, ORCID: 0000-0003-0276-4517 <itsykson@yandex.ru>
Университет ИТМО,
Россия, 197101, Санкт-Петербург, Кронверкский проспект, д.49, л. А.

Аннотация. В последние годы на рынке мобильных операционных систем наблюдается тенденция создания альтернативных платформ, ориентированных на собственные языки программирования и инструменты разработки. Компании стремятся снизить зависимость от существующих решений на базе операционных систем (ОС) Android и iOS, развивая собственные экосистемы. Одним из примеров подобного подхода является система HarmonyOS NEXT компании Huawei, которая полностью абстрагировалась от ОС Android и функционирует в собственной экосистеме, включающей в свой состав оригинальные языки программирования, собственные среды разработки и независимый магазин приложений. Платой за независимость на первом этапе является является бедное наполнение пользовательскими приложениями: для Android и iOS количество приложений измеряется миллионами, а для Harmony OS NEXT – несколькими тысячами. Очевидно, что проблема наполнения экосистемы приложениями является очень актуальной задачей. В данной работе предлагается подход к наполнению экосистемы HarmonyOS NEXT путем автоматизированного портирования Android-приложений. Подход заключается в систематической трансляции компонентов исходного Kotlin-приложения в соответствующие компоненты на языке программирования ArkTS, являющимся нативным для клиентских приложений в HarmonyOS NEXT, с сохранением исходной логики и структуры программы. Самым сложным для портирования компонентом являются библиотеки. Для их переноса в работе применяется подход автоматизации портирования библиотек на основе спецификаций, заданных на языке LibSL. Видимое поведение библиотек Android и HarmonyOS NEXT описывается в виде поведенческих спецификаций, на основе которых проверяется логическая совместимость двух библиотек и, в случае успеха, синтезируется цепочка вызовов ArkTS-библиотеки, заменяющая исходный вызов библиотеки Kotlin. Проверка совместимости и синтез осуществляются с привлечением SMT-солвера. Предложенный подход был реализован в виде прототипа инструмента, который был апробирован на простых Android-приложениях. Тестируемые приложения были успешно портированы в эквивалентные по поведению приложения ArkTS.

Ключевые слова: операционная система HarmonyOS NEXT; язык программирования ArkTS; язык программирования Kotlin; портирование исходного кода; формальные спецификации; язык спецификации LibSL; решатель SMT Solver.

Для цитирования: Миленин И.А., Ицыксон В.М. Автоматизация портирования Android-приложений в экосистему HarmonyOS NEXT с использованием формальных спецификаций. Труды ИСП РАН, том 38, вып. 3, часть 4, 2026 г., стр. 83–100. DOI: 10.15514/ISPRAS–2026–38(3)–48.

Automation of porting Android applications to the HarmonyOS NEXT ecosystem using formal specifications

I.A. Milenin, ORCID: 0009-0000-6151-4038<svakun@gmail.com>
V.M. Itsykson, ORCID: 0000-0003-0276-4517 <itsykson@yandex.ru>
ITMO University,
bldg. A, 49, Kronverksky Ave., St. Petersburg, 197101, Russia.

Abstract. In recent years, there has been a trend in the mobile operating system market towards the creation of alternative platforms focused on proprietary programming languages and development tools. Companies are striving to reduce their dependence on existing solutions based on Android and iOS by developing their own ecosystems. One example of this approach is Huawei's HarmonyOS NEXT, which is completely abstracted from the Android OS and operates within its own ecosystem, including original programming languages, development environments, and an independent app store. The price for this independence in the initial phase is the lack of user applications: while Android and iOS have millions of applications, HarmonyOS NEXT has only a few thousand. Clearly, the issue of populating the ecosystem with applications is a very relevant task. This paper proposes an approach to populating the HarmonyOS NEXT ecosystem through automated porting of Android applications. The approach involves the systematic translation of components from the original Kotlin application into corresponding ArkTS components, which are native to client applications in HarmonyOS NEXT, while preserving the original logic and program structure. The most challenging component to port is libraries. To address this, the paper uses an automation approach for porting libraries based on specifications defined in the LibSL language. The observable behavior of Android and HarmonyOS NEXT libraries is described using behavioral specifications, which are then used to verify the logical compatibility of the two libraries. If compatible, an ArkTS library call chain is synthesized to replace the original Kotlin library call. Compatibility checking and synthesis are performed using an SMT solver. The proposed approach was implemented as a prototype tool, which was tested on simple Android applications. The tested applications were successfully ported to equivalent ArkTS applications with identical behavior.

Keywords: HarmonyOS NEXT; ArkTS; Kotlin; source code porting; formal specifications; LibSL; SMT Solver.

For citation: Milenin I.A., Itsykson V.M. Automation of porting Android applications to the HarmonyOS NEXT ecosystem using formal specifications. Trudy ISP RAN/Proc. ISP RAS, vol. 38, issue 3, part 4, 2026, pp. 83-100 (in Russian). DOI: 10.15514/ISPRAS-2026-38(3)-48.

1. Введение

С развитием индустрии мобильных технологий наблюдается тенденция к созданию крупными игроками собственных операционных систем, ориентированных на независимую экосистему и использующих оригинальные языки программирования. Крупные компании стремятся снизить зависимость от сторонних решений, так как любые зависимости приводят к снижению технологической автономности. В результате компания остаётся уязвимой перед возможными ограничениями или прекращением доступа к используемым технологиям.

Для обеспечения собственной независимости в компании Huawei была разработана оригинальная операционная система HarmonyOS NEXT, разработка для которой ведется на новых языках программирования: ArkTS и Cangjie. Построение новой экосистемы позволило полностью отказаться от зависимости к ОС Android. Однако это привело к невозможности запуска Android-приложений.

В связи с вышесказанным возникла необходимость быстрого наполнения экосистемы новой операционной системы востребованными пользовательскими приложениями. Очевидно, что спустя определенное время многие разработчики мобильных приложений начнут выпускать версии своих приложений и для Harmony OS NEXT, как это было, когда на рынке появился Android, но это длительный многолетний процесс, в то время как приложения нужны уже сейчас. Поэтому, актуальной задачей в настоящее время является разработка инструментов

портирования, которые обеспечат автоматическую миграцию приложений из существующих экосистем в новую.

В данной работе представлен подход, обеспечивающий автоматизацию переноса Android-приложений, написанных на языке Kotlin, в среду Harmony OS NEXT. Подход обеспечивает преобразование кода Kotlin-приложений, в эквивалентный код на языке ArkTS, сохраняя архитектуру и логику работы программы.

При переносе приложений из одной платформы на другую недостаточно обычной трансляции исходного кода с одного языка на другой (транспиляции). Довольно весомая часть функциональности экосистемы Android использует классы и методы из стандартных библиотек Kotlin, а также фреймворки, которые тесно завязаны на использовании методов из Android SDK. Для успешной адаптации проекта под новую платформу нужно заменить использование подобных классов и библиотек на эквивалентные им решения в экосистеме HarmonyOS NEXT. В данной работе мы используем формальные спецификации внешних библиотек для автоматизации процесса трансляции. Разрабатываются поведенческие спецификации исходных библиотек, написанных на Kotlin и Java, а также спецификации библиотек целевой платформы, написанных на языке ArkTS. В качестве языка описания спецификаций выбран язык LibSL, так как он специально был разработан для описания поведения библиотек и поддерживает работу с большинство современных императивных языков программирования. Метод проверки совместимости определяет поведенческую совместимость двух библиотек. В случае успеха применяется метод автоматической замены исходной библиотеки на целевую с сохранением семантики поведения программы, основанный на преобразовании семантических трасс в цепочки формул, для разрешения которых используются решатели SMT-солвер.

Разработанный подход реализован в виде прототипа инструмента портирования, цель которого показать принципиальную возможность автоматизация процесса трансляции на основе спецификаций.

Статья организована следующим образом. Раздел 2 посвящен общему описанию подхода и архитектуры инструмента портирования. В разделе 3 описываются механизмы преобразования основных конструкций исходного кода Kotlin-программ в ArkTS. В разделе 4 приводится методы преобразования специфических конструкций. Раздел 5 посвящен портированию используемого библиотечного окружения на основе формальных спецификаций. В шестом разделе кратко описывается опыт применения инструментов к портированию простых Android-приложений. Раздел 7 посвящен анализу имеющихся на рынке инструментов, пригодных для решения задачи портирования. В заключение делаются выводы о применимости подхода и определяются направления дальнейших исследований.

2. Предлагаемый подход

Процесс автоматизации портирования Android-приложений в экосистему HarmonyOS NEXT основан на идее системного преобразования исходного кода с языка Kotlin в язык ArkTS, с сохранением логики работы программы и корректной заменой используемого библиотечного окружения.

Портирование включает три ключевые составляющие:

- Преобразование исходного текста программы, то есть перевод конструкций языка Kotlin в эквивалентные конструкции ArkTS;
- Замена библиотек и SDK, принадлежащих экосистеме Android/Kotlin, на функционально эквивалентные элементы в экосистеме HarmonyOS NEXT;
- Портирование описанного декларативно пользовательского интерфейса приложения Jetpack Compose в ArkUI.

Такой подход позволяет обеспечить не только синтаксическую корректность результирующего кода, но и семантическое соответствие поведения программы в новой среде. В рамках данной статьи мы сосредоточимся на первых двух составляющих, автоматизация портирования пользовательского интерфейса – самостоятельная сложная наукоемкая задача, которая является предметом отдельной статьи.

2.1 Преобразование исходного кода Kotlin в ArkTS

Для реализации первой части – перевода исходного кода – в инструменте используется концепция промежуточных представлений (IR, Intermediate Representation).

Идея заключается в том, чтобы отделить анализ исходного кода от генерации целевого, создав универсальный слой, описывающий программу на абстрактном уровне.

Для этого применяются расширенные абстрактные синтаксические деревья (AST):

- для Kotlin используется модель PSI (Program Structure Interface), формируемая библиотекой Kotlin Analysis API;
- для ArkTS формируется собственное промежуточное представление – ArkTS IR, которое отражает структуру программы в терминах элементов языка ArkTS.

Исходный код Android-приложения преобразуется в представление PSI, откуда позднее извлекается полная структура проекта: классы, функции, объекты, свойства, модификаторы доступа и т.д. Каждый элемент исходного кода обрабатывается отдельно и отображается в соответствующий элемент ArkTS IR.

Для конструкций языка Kotlin, которые имеют прямые аналоги в языке ArkTS, используется шаблонное преобразование.

На этом этапе выполняется:

- Замена синтаксических ключевых слов и деклараций (fun заменяется на function, val заменяется на const, when заменяется на switch, object заменяется на namespace, т.п);
- Коррекция модификаторов доступа и видимости. В Kotlin имеется модификатор internal, отсутствующий в ArkTS, из-за этого он транслируется в public, так как область видимости модулей в ArkTS определяется иначе. Оставшиеся модификаторы видимости (public, private, protected) сохраняются без изменений
- Коррекция структуры классов и их элементов в соответствии со спецификацией ArkTS. Объявления class переносятся напрямую, включая первичные и вторичные конструкторы.
- Перенос базовых типов. Типы, совпадающие по назначению, преобразуются напрямую: Int и Double преобразуется в number, так как множество класса number в ArkTS объединяет целочисленные и вещественные значения. Тип Boolean сохраняется без изменений. String также транслируется напрямую, поскольку представление строк в обоих языках является совместимым.

Особое внимание уделяется nullable-типам: если в Kotlin переменная объявлена с ?, то в ArkTS она получает объединённый тип T | null, что эквивалентно по смыслу обработке отсутствующего значения.

Сложность при преобразовании возникает, когда исходная конструкция на языке Kotlin не имеет точного соответствия в языке ArkTS. К таким конструкциям относятся data class, sealed class, companion object, inline-функции и другие элементы Kotlin, которые не были поддержаны в целевом языке программирования.

Для таких элементов инструмент применяет правила трансляции, основанные на их семантике, а не только синтаксисе:

- при обнаружении data class генерируются явные реализации методов equals(), toString() и copy(), отсутствующих в ArkTS;
- sealed class преобразуется в набор вложенных классов с ограниченным наследованием;
- companion object отображается как пространство имён (namespace) с эквивалентными свойствами и функциями.

Таким образом, первая часть инструмента обеспечивает корректное отображение структуры программы на уровне исходного языка, формируя эквивалентное ArkTS IR-представление. Такие шаги помогают уменьшить количество измененных вызовов методов в исходном коде, что упрощает перевод приложения на целевую платформу.

2.2 Замена библиотек и SDK

Вторая часть наиболее сложна в трансляции, так как, в отличие от первой части, где список конструкция языка конечен и их синтаксис и семантика четко определены и закреплены в спецификации языка программирования, число библиотек потенциально бесконечно и их семантика определяется исключительно автором библиотеки. Это означает, что ни один инструмент не может заранее знать семантику всех существующих библиотек – она должна быть явно описана.

Для решения этой проблемы в данной работе применяется подход к автоматизации портирования библиотек на новое окружение, разработанный авторами ранее ([1], [2]).

Его ключевая идея заключается в использовании формальных спецификаций, описывающих поведение библиотек обеих платформ – исходной (Kotlin/Android) и целевой (ArkTS/HarmonyOS NEXT).

Специалист по портированию разрабатывает две группы спецификаций:

- исходные спецификации, описывающие поведение библиотек Kotlin и Android SDK, используемых в анализируемом приложении;
- целевые спецификации, определяющие эквивалентное поведение библиотек ArkTS и HarmonyOS.

При портировании вызовов функции производится проверка, имеются ли её описания в наборе спецификаций. Если такая спецификация найдена, то инструмент извлекает формальное описание поведения функции, включая ее предусловия, постусловия и побочные эффекты. Эти элементы функции трансформируются в трассы исполнения, определяющие ее поведение.

Далее выполняется поиск описанных выше трасс исполнения функции. Такая трасса составляется на основе полученных спецификаций как для исходных, так и для целевых методов. На основе трассы формируется формальная модель, которая отражает зависимость между входными и выходными параметрами, промежуточными вычислениями и возвращаемыми значениями. Такая модель представляет из себя абстрактное математическое описание логики работы функции.

На основе созданной модели производится сопоставление формальных моделей исходной и целевой функций. Для проверки совместимости применяется алгоритм, активно использующий вызовы SMT-солвера, с помощью которого сопоставляются логические выражения, описывающее поведение обеих функций, и определяется факт их эквивалентности.

Если алгоритм подтверждает, что модели демонстрируют сопоставимое поведение, то на основе соответствующей трассы производится генерация промежуточного представления функции-посредника, на вызов которой будет заменен вызов исходный функции. В противном случае вызов функции останется неизменным.

После завершения формирования промежуточного представления всего исходного кода на его основе генерируется целевой ArkTS код. Все описанные выше шаги показаны на рис. 1.

Важно отметить, что хотя работа по созданию спецификаций довольна сложная, она для каждой библиотеки выполняется однократно, а использовать эти спецификации можно будет многократно при портировании различных приложений.

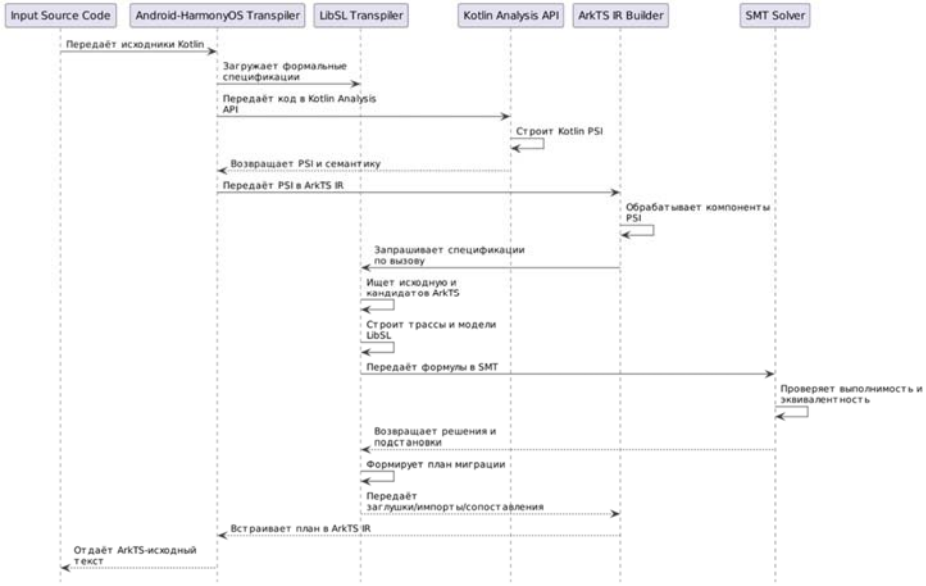

Рис. 1. Диаграмма последовательности транспиляции Kotlin в ArkTS.
Fig. 1. Sequence diagram of Kotlin-to-ArkTS transpilation.

3. Портирование стандартных конструкций

Одной из базовых задач портирования Android-приложений является преобразование языковых конструкция языка Kotlin в семантически эквивалентные конструкции языка ArkTS. Обычно такие преобразования делают не напрямую, а с использованием различных промежуточных представлений программы. Основное требование к таким представлениям – адекватное отображение синтаксиса и возможность отображения части семантики. В платформах для разработки компании Jetbrains таким представлением является PSI (Program Structure Interface).

Возможным альтернативным подходом к портированию могло бы являться использование многошаговой трансляции с промежуточным преобразованием Kotlin-кода в Java, с последующим применением существующих методов переноса Java-программ. Однако такой подход приводит к потере части структурной и семантической информации программы, связанной с использованием высокоуровневых языковых абстракций. Кроме того, последовательная цепочка трансляций увеличивает количество промежуточных представлений и трансляционных артефактов, что усложняет дальнейшие преобразования и анализ программы.

В рамках данной работы рассматривается прямое портирование программ с использованием PSI как промежуточного представления, что позволяет оперировать формализованной моделью программы и выполнять преобразования без введения дополнительных промежуточных этапов. На выходе подобный подход позволяет получить целевую

программу, семантически эквивалентную исходной, и обеспечить более предсказуемый процесс её сопровождения в экосистеме HarmonyOS NEXT.

Для построения PSI по программе на языке Kotlin используется библиотека Kotlin Analysis API [3], которая формирует PSI, соответствующий исходной программе. После формирования PSI дерева осуществляется семантический анализ, который определяет связи между элементами программы, типами выражениями, компонентами. Результатом работы является формализованная модель, которая объединяет в себе синтаксическое и семантическое представление программы. В дальнейшем эта модель используется для проведения всех преобразований языковых конструкций и формирования ArkTS IR – промежуточного представления, используемого для генерации ArkTS-программ.

3.1 Анализ синтаксического дерева и выполнение преобразований

Для выполнения преобразований производится обход PSI, построенного ранее. Каждая из вершин абстрактного синтаксического дерева соответствует какому-либо элементу Kotlin-кода. Корневым элементом всех PSI-структур является класс KtElement, от которого наследуются различные сущности: KtFile, KtClass, KtNamedFunction, KtProperty, KtExpression и другие. Алгоритм последовательно обходит дерево PSI, начиная от корневого узла KtFile, который представляет собой отдельный исходный файл Kotlin-программы.

Для каждого типа узла используется специализированный обработчик-преобразователь. Примеры таких преобразователей приведены в табл. 1. Например, KotlinClassTranspiler отвечает за преобразование классов и объектов, KotlinTopLevelFunctionTranspiler отвечает за функции верхнего уровня, а KotlinExpressionTranspiler осуществляет портирование выражений и операторов.

3.2 Формирование промежуточного представления ArkTS IR

Во время обработки PSI формируются аналогичные конструкции, представляющие собой элементы внутреннего промежуточного представления – ArkTS IR. ArkTS IR не повторяет структуру Kotlin-кода напрямую, а формирует новую модель программы в терминах ArkTS [4], где каждый элемент отражает семантику исходного узла, но уже в синтаксических и типовых рамках целевого языка.

В рамках реализации подобного промежуточного представления каждая из вершин является наследником базового элемента-интерфейса ArkTsNode, который определяет общий контракт для всех элементов абстрактного синтаксического дерева ArkTS. Каждый узел, реализующий интерфейс ArkTsNode, обязан уметь восстанавливать собственное представление в виде исходного кода ArkTS с помощью метода render(). Благодаря этому каждый узел IR способен самостоятельно формировать свою текстовую форму.

Стоит отметить, что из-за того, что исходный и целевой языки обладают различными синтаксическими и семантическими особенностями, преобразование между ними не может быть прямым или симметричным. Ввиду этого при построении ArkTS IR в инструменте реализована система контекстных преобразований, которая адаптирует элементы исходного кода с учетом различий языков:

- ключевые слова val и var преобразуются в слова const и let;
- выражения Kotlin с использованием when и is транслируются в switch и instanceof;
- функции верхнего уровня в классах помещаются в namespace, соответствующий имени класса;
- модификаторы доступа и области видимости корректируются в соответствии с правилами ArkTS;
- и т.п.

Табл. 1. Основные преобразователи синтаксических элементов Kotlin.
Table 1. Primary mappers for Kotlin syntactic elements.

Наименование преобразователя	Обрабатываемый элемент	Выполняемые задачи
KotlinCodeTranspilerImpl	KtFile	Собирает состояние анализа и рендерит итоговый ArkTS-код с импортами и сгенерированными функциями.
KotlinDeclarationTranspiler	Верхнеуровневая KtDeclaration	Делегирует обработку функции, класса и объекта специализированным транспилятором.
KotlinClassTranspiler	KtClass	Определяет тип класса и отправляет его в нужный класс-транслятор (interface/enum/sealed/data/обычный).
KotlinObjectTranspiler	KtObjectDeclaration	Превращает объекты в ArkTS namespace с функциями, свойствами и вложенными декларациями.
FunctionCoreMapper	KtNamedFunction	Извлекает имя, параметры, типы и тело функции как основу для дальнейших адаптеров.
BaseKotlinClassMapper	Обычный KtClass	Строит базовое объявление ArkTS-класса: конструкторы, свойства, функции, наследование.
SealedClassMapper	KtClass с sealed + вложенные элементы	Оборачивает sealed-класс в ArkTS sealed-декларацию и namespace, добавляя экспорт подклассов и объектов.
ExpressionMapper	Любой KtExpression	Конвертирует выражения Kotlin в соответствующие узлы ArkTS.
KotlinToArkTsTypeMapper	KotlinType	Переводит типы Kotlin в ArkTS, учитывая nullability и generics.

4. Преобразование Kotlin PSI в ArkTS IR

Из-за того, что не для всех Kotlin конструкций есть прямые аналоги в языке программирования ArkTS требуется реализация таких механизмов, которые обеспечивали бы эквивалентное поведение программы на целевой платформе.

4.1 Портирование data-классов

В языке ArkTS отсутствует прямой аналог data-классов, для его переноса необходима специальная адаптация, которая позволит не просто синтаксически преобразовать код, а создать корректное ArkTS-представление программы, сохраняющее ее логику и архитектурные зависимости. В языке Kotlin data-классы автоматически генерируют служебные методы (equals, toString, copy), чего нет в ArkTS (за исключением toString). Поэтому при обработке таких классов инструмент самостоятельно формирует недостающие методы на уровне промежуточного представления (IR).

Сначала на основе базового отображения (BaseKotlinClassMapper) формируется базовая ArkTS-декларация класса (ArkTsClassDeclaration). Затем преобразователь анализирует, были ли реализованы методы toString, equals и copy. Если они отсутствуют, то автоматически генерируется их реализация. Так метод equals генерируется как логическое выражение, сравнивающее все параметры конструктора через строгие операторы === и instanceof. Метод toString же формирует строковое представление через конкатенацию имен и значений

параметров класса. Итоговая промежуточное представление подобного класса объединяет в себе все сгенерированные методы и возвращает его.

На рис. 2 приведен пример data-класса на языке Kotlin, а на рис. 3 сгенерированный эквивалентный ему код на языке ArkTS.

```
data class Foo(val a: Int, val b: String)
```

Рис. 2. Пример исходного data-класса.
Fig. 2. Example of a source data class.

```
export class Foo {
  public readonly a: number;

  public readonly b: string;

  constructor(a: number, b: string) {
    this.a = a
    this.b = b
  }

  public equals(other: object | null): boolean {
    return other === this || other instanceof Foo && this.a === other.a && this.b === other.b
  }

  public toString(): string {
    return "Foo(" + "a=" + this.a + ", " + "b=" + this.b + ")"
  }

  public copy(a: number = this.a, b: string = this.b): Foo {
    return new Foo(a, b)
  }
}
```

Рис. 3. Пример сгенерированного аналога data-класса в ArkTS.
Fig. 3. Example of a generated data class analogue in ArkTS.

4.2 Портирование sealed-классов

Похожие принципы применяется при обработке sealed-классов. Подобная конструкция также отсутствуют в ArkTS, так что требуется генерировать такую реализацию, которая обеспечивала бы сохранение семантики.

Портирование sealed-классов происходит следующим образом:

1) на основе базового отображения (BaseKotlinClassMapper) формируется базовая ArkTS-декларация класса (ArkTsClassDeclaration);
2) происходит анализ всех вложенных классов и объектов, находящихся внутри sealed-класса;
3) создается абстрактный класс, от которого в дальнейшем будут унаследованы все вложенные объекты. при необходимости он заполняется общими методами и параметрами, которые возможно будет переопределить;
4) создается пространство имен (namespace), наименование которого совпадает с изначальным sealed-классом и созданным на предыдущем шаге абстрактным классом, данное пространство имен позволит сохранить иерархию и гарантировать, что все наследники объявлены в одном месте;

Все классы, являющиеся наследниками исходного sealed-класса, просто портируются, как и другие классы в рамках реализуемого инструмента. Однако для объектов (object), которые

являются наследниками исходного sealed-класса, происходит создание специальной структуры, включающей как декларацию класса, так и экземпляр этого класса. Для каждого такого объекта формируется новый класс с суффиксом Class, содержащий возможные свойства и функции, после чего создаётся единственный экземпляр этого класса внутри пространства имён. Этот экземпляр экспортируется под исходным именем объекта и становится доступным для использования как константа.

На рис. 4 приведен пример sealed-класса на языке Kotlin (рис. 4). Сгенерированный на его основе аналог на языке ArkTS приведен на рис. 5.

Таким образом, формирование промежуточного представления ArkTS IR завершает основной этап преобразования исходного кода. На этом уровне достигается не только синтаксическая конвертация конструкций, но и семантическое выравнивание логики программы между Kotlin и ArkTS, что позволяет портировать исходный код с сохранением его основного повеления.

```
sealed class Result {
    data class Success(val data: String) : Result()
    object Error : Result()
}
```

Рис. 4. Пример исходного sealed-класса
Fig. 4. Example of a source sealed class.

```
export abstract class Result {
}
export namespace Result {
  export class Success extends Result {
    public readonly data: string;

    constructor(data: string) {
      super()
      this.data = data
    }

    public equals(other: object | null): boolean {
      return other === this || other instanceof Success && this.data === other.data
    }

    public toString(): string {
      return "Success(" + "data=" + this.data + ")"
    }

    public copy(data: string = this.data): Success {
      return new Success(data)
    }
  }

  class ErrorClass extends Result {
    constructor() {
      super()
    }
  }

  export const Error: Result = new ErrorClass();
}
```

Рис. 5. Пример сгенерированного аналога sealed-класса в ArkTS.
Fig. 5. Example of a generated sealed class analogue in ArkTS.

5. Формальные спецификации и проверка корректности преобразований

На этапе преобразование Kotlin PSI в ArkTS IR решается задача корректного синтаксического отображения конструкций из языка Kotlin в язык ArkTS. Однако в коде могут вызываться функции или классы, которые принадлежат стандартным библиотекам Kotlin или Android SDK и не имеют прямых аналогов в экосистеме HarmonyOS, либо синтаксис этих аналогов отличается, из-за чего портирование невозможно выполнить простым сопоставлением исходных и целевых конструкций. В таких случаях требуется дополнительный уровень анализа, обеспечивающий семантическое соответствие между элементами двух платформ.

Чтобы решить эту задачу, в инструменте реализован механизм формальных спецификаций, позволяющий формально описывать видимое поведение функций и классов, независимо от их конкретной реализации. Формальное сравнение видимых поведений, описываемых спецификациями, делает возможным автоматическое сравнение поведения библиотечных элементов Kotlin и ArkTS при портировании.

Подход основан на использовании языка формальных спецификаций LibSL (Library Specification Language) [5]. Этот язык позволяет декларативно описывать поведение библиотечных функций, классов и их взаимодействие с состоянием программы. Он ориентирован на формальное моделирование, а не на реализацию, что делает возможным анализ и сравнение поведения библиотек разных платформ.

При инициализации инструмент портирования производит анализ всех получаемых на вход спецификаций. Спецификации, в заголовке файла которых указан язык language "arkts" считаются целевыми. Для всех спецификаций формируется абстрактное синтаксическое дерево с помощью библиотеки libsl-parser. Далее на основе полученного синтаксического дерева формируется трасса исполнения – последовательность действий, моделирующих выполнение функции в терминах операций над состоянием.

На следующем этапе инструмент формирует формальную модель поведения функции – систему логических выражений, описывающих зависимости между входными параметрами, промежуточными вычислениями и результатом. Такие модели для исходных и целевых функций сравниваются при помощи SMT-солвера [6].

SMT-солвер выполняет проверку эквивалентности формальных моделей, определяя, приводят ли две функции – исходная (на Kotlin) и целевая (на ArkTS) – к одинаковым результатам при совпадающих входных данных. Если система логических формул, описывающая их поведение, оказывается выполнимой одновременно (то есть не содержит противоречий), функции признаются семантически эквивалентными.

Для реализации данной проверки используется комбинация языка спецификаций LibSL для описания поведения библиотек и платформы KSMT, предоставляющей собой интеллектуальный интерфейс к SMT-солверам и механизмы символьного исполнения. Подобные подходы к проверке функциональной эквивалентности уже зарекомендовали себя в области автоматизированной генерации тестов и верификации программ [7]. Также подобный подход показал положительный результат в рамках исследования возможности автоматизации портирования библиотек языка программирования Java. В разработанном инструменте механизм формальной проверки был адаптирован и расширен для поддержки анализа библиотек Kotlin и их сопоставления с аналогами на ArkTS.

Когда возникает необходимость определить, существует ли эквивалент функции из Kotlin-библиотеки в ArkTS, происходят следующие действия:

1) Проверяется, объявлена ли соответствующая функция пользователем в рамках переданного им исходного кода. Если функция есть, то вызов остается неизменным. В противном случае инструмент прибегает к попытке заменить данную функцию на функцию, основанную на формальных спецификациях.

2) Происходит поиск подходящей исходной функции в формальных спецификациях по её полному имени и сигнатуре (типам параметров и результирующему типу).
3) Для исходной и каждой из целевых функций строятся трассы исполнения.
4) Для всех трасс формируется система логических уравнений, отражающая зависимости между аргументами, промежуточными вычислениями и результатами выполнения. Каждое семантическое действие кодируется в виде логического выражения, а равенства между действиями, результатами и аргументами приводятся к каноническому виду.
5) Полученная система уравнений преобразуется в логическую формулу для SMT-решателя. Решатель проверяет корректность и выполнимость этой системы. Если формула оказывается неразрешимой (UNSAT) либо результат проверки является неопределённым (UNKNOWN), считается, что семантическое соответствие между функциями отсутствует. В этом случае инструмент выводит в консоль предупреждение о невозможности сопоставления соответствующего библиотечного метода по формальным спецификациям. Для обеспечения завершенности процесса портирования в целевой программе генерируется узел ArkTS IR, сохраняющий исходное имя функции и набор её аргументов, что позволяет явно зафиксировать не поддержанный вызов и при необходимости выполнить его последующую ручную обработку
6) Для всех возможных трасс исходной функции повторяется процесс сопоставления. При отсутствии решения хотя бы для одной трассы делается вывод о несовместимости библиотек. В противном случае фиксируется набор эквивалентных трасс, описывающих, как исходная функция может быть выражена через вызовы функций целевой библиотеки.
7) На заключительном этапе из найденных решений формируется промежуточное представления сгенерированной функции с префиксом «generated_». При необходимости добавляется import для целевой библиотеки.

Такой подход позволяет явно сопоставлять функции из исходных и целевых библиотек, что позволяет более точно производить преобразования исходного кода в целевой, как показано на рис. 6 и 7.

6. Результат применения подхода

В ходе работы был разработан инструмент, реализующий предложенный подход, и позволяющий автоматически переносить одномодульные Android-приложения в экосистему операционной системы HarmonyOS. Благодаря данному инструменту было портировано несколько синтезированных мобильных приложений, разработанных самостоятельно для тестирования инструмента.

Также для оценки практической применимости предложенного подхода был проведён эксперимент по автоматизированному портированию реального Android-приложения, исходный код которого был взят из открытых источников, в экосистему HarmonyOS NEXT.

В качестве тестового объекта было выбрано приложение, содержащее пользовательский интерфейс, вычислительную бизнес-логику и обращения к стандартным библиотекам языка Kotlin. Размер приложения позволяет рассматривать его как репрезентативный пример типичного прикладного Android-приложения, при этом сложность его структуры достаточна для демонстрации работы всех ключевых компонентов предложенного инструмента.

Количественные характеристики исходного и целевого приложений приведены в табл. 2. После автоматического портирования количество классов в целевом приложении увеличилось до 14, а объём бизнес-логики – до 270 строк. Увеличение объёма кода обусловлено различиями в языковых и архитектурных абстракциях Kotlin и ArkTS. В

частности, отсутствие прямых аналогов некоторых конструкций Kotlin в ArkTS приводит к необходимости генерации вспомогательных классов и методов, а также к более явному описанию поведения, которое в Kotlin задаётся декларативно. Таким образом, рост объёма кода является ожидаемым и не связан с дублированием бизнес-логики.

kotlin-math.lsl

```
libsl "1.0.0";
library `KotlinMath`
    version "1.0.0";

typealias int = int32;
typealias double = float64;

define action SQRT(x: double): double;
define action POW(x: double, y: double): double;

type kotlin.math { }

automaton kotlin.math: kotlin.math {

    fun hypot(x: double, y: double): double {
        var x2: double = action POW(x, 2.0);
        var y2: double = action POW(y, 2.0);
        result = action SQRT(x2 + y2);
    }

}
```

arkts-math.lsl

```
libsl "1.0.0";
library `ArkTsMath`
    version "1.0.0"
    language "arkts";

typealias int = int32;
typealias double = float64;

define action SQRT(x: double): double;
define action POW(x: double, y: double): double;

type Math { }

automaton Math: Math {

    fun sqrt(x: double): double {
    requires x >= 0.0;
        result = action SQRT(x);
    }

    fun pow(x: double, y: double): double {
        result = action POW(x, y);
    }

}
```

Рис. 6. Формальные спецификации функции hypot в Kotlin и эквивалентных действий в ArkTS на языке LibSL.
Fig. 6. Formal specifications of the hypot function in Kotlin and its equivalent operations in ArkTS using the LibSL language.

```
fun distance(x1: Double, y1: Double, x2: Double, y2: Double): Double {
    return kotlin.math.hypot(x2 - x1, y2 - y1)
}
<END>
=== Результат ===
// File: Source.kt

export function distance(x1: number, y1: number, x2: number, y2: number): number {
    return generated_hypot(x2 - x1, y2 - y1)
}
function generated_hypot(x: number, y: number): number {
    const result0: number = Math.pow(x, 2.0);
    const result1: number = Math.pow(y, 2.0);
    const result2: number = Math.sqrt(result0 + result1);
    return result2
}

=== Конец результата ===
```

Рис. 7. Пример результата автоматического преобразования функции distance() из Kotlin в ArkTS с генерацией эквивалентной функции generated_hypot().
Fig. 7. Example of automatic translation of the distance() function from Kotlin to ArkTS with generation of the equivalent generated_hypot() function.

Табл. 2. Результаты автоматического портирования Android-приложения в HarmonyOS NEXT.
Table 2. Results of automated porting of an Android application to HarmonyOS NEXT.

	Android	HarmonyOS NEXT
Количество классов	6	14
Объем бизнес-логики	120 строк	270 строк
Количество методов, описанных через формальные спецификации	6	6
Объем формальных спецификаций	67 строк	64 строки
Время автоматического портирования	2,5 с	

Бизнес-логика приложения была перенесена полностью в автоматическом режиме. Пользовательские функции и вычислительные выражения транслировались посредством анализа PSI-дерева Kotlin, построения промежуточного представления ArkTS IR и последующей генерации целевого кода. Полученный ArkTS-код компилируется и воспроизводит семантику исходной программы, что подтверждает корректность выбранной схемы трансляции пользовательского кода.

В рассматриваемом приложении использовалось шесть библиотечных методов, не имеющих прямых аналогов в HarmonyOS NEXT. Для всех этих методов были подготовлены формальные спецификации поведения на языке LibSL как для исходных, так и для целевых библиотек. На основе спецификаций строились модели поведения функций, после чего выполнялась проверка эквивалентности семантических трасс с использованием SMT-решателя. Во всех случаях эквивалентность поведения была подтверждена, что позволило автоматически заменить вызовы библиотек без изменения логики приложения.

При переносе были успешно портированы все элементы, относящиеся к вычислительной логике и модельному слою приложения, включая структуру sealed-классов и data-классов. При портировании была проигнорирована часть кодовой базы, отвечающая за визуальное представление интерфейса. Это обусловлено тем, что подходы к портированию декларативных UI фреймворков выходят за рамки данного исследования. Отдельно стоит отметить, что благодаря использованию формальных спецификаций стандартных библиотек удалось корректно перенести вызовы Kotlin-методов, сохранив их семантику и поведение в среде выполнения ArkTS.

Полученные результаты демонстрируют реализуемость предложенного подхода и его применимость не только к синтетическим примерам, но и к реальному программному коду. Предложенная технология позволяет автоматически переносить значительную часть Android-приложений в экосистему HarmonyOS NEXT с сохранением семантики бизнес-логики и корректной заменой библиотечных вызовов. Это подтверждает целесообразность дальнейшего развития инструмента и расширения экспериментальной базы за счёт более крупных и сложных проектов.

7. Анализ существующих подходов к портированию приложений

Проблема переноса Android-приложений в экосистему HarmonyOS NEXT стала особенно актуальной после отказа Huawei от поддержки Android Runtime (AOSP). Теперь разработчикам необходимо создавать приложения заново на языке ArkTS, что делает задачу автоматического портирования чрезвычайно востребованной.

Существующие подходы к решению данной задачи можно поделить на две основные группы:

- решения, основанные на статическом анализе и формальных правилах;

- решения, применяющие нейросетевые модели для синтаксической и семантической реконструкции кода.

7.1 Инструменты, основанные на статическом анализе и формальных правилах

К данной категории относятся подходы, опирающиеся на статический анализ и формальное моделирование поведения программ.

На текущий момент отсутствуют полноценные транспиляторы, выполняющие прямое преобразование Kotlin-кода в ArkTS. Несмотря на частичное синтаксическое сходство ArkTS с TypeScript, данный язык имеет собственную модель выполнения, систему типизации и механизм сборки через Ark Compiler, что делает невозможным применение существующих решений без адаптации.

Ближайшим аналогом можно считать официальный компилятор Kotlin/JS [8] от компании JetBrains, который обеспечивает трансляцию Kotlin в JavaScript с генерацией деклараций, совместимых с TypeScript. Однако данный механизм ориентирован исключительно на веб-платформы и не учитывает архитектурные особенности ArkTS, включая декларативную модель интерфейсов ArkUI, взаимодействие с системными сервисами HarmonyOS и особенности асинхронного выполнения.

Можно констатировать, что не существует инструментов, формально транслирующих Android-приложения в экосистему Harmony OS NEXT.

7.2 Инструменты, использующие модели машинного обучения

Развитие методов машинного обучения и больших языковых моделей (LLM) привело к появлению инструментов, выполняющих автоматизированное портирование кода и интерфейсов при помощи нейросетевых архитектур.

Одним из таких примеров является система UITrans – многоагентное решение, представленное в исследовании UITrans: Seamless UI Translation from Android to HarmonyOS [9]. Данный инструмент осуществляет анализ XML-разметки пользовательского интерфейса Android, формирование промежуточного описания структуры интерфейса, а в заключении генерирует эквивалентных компонентов ArkUI на языке ArkTS с учётом визуальных и поведенческих зависимостей. Проблемой данного инструмента является то, что он ориентирован на устаревший подход к декларированию пользовательского интерфейса – XML-разметки, когда на данный момент современным стандартом написания пользовательских интерфейсов является технология Jetpack Compose.

Также существует подход к портированию кода прикладной логики и системных конструкций языка Java. В работе LLM-Based Java Concurrent Program to ArkTS Translation [10] рассматривается задача автоматического преобразования Java-программ в эквивалентный код на ArkTS, ориентированный на выполнение в среде HarmonyOS NEXT. В данном исследовании авторы также опираются на большие языковые модели, способные анализировать исходный Java-код, выделять семантические зависимости между методами и объектами, и затем формировать аналоги в ArkTS. Однако на данный момент основным языком для разработки клиентских приложений под операционную систему Android является Kotlin, а не Java В связи с этим новые приложения пишутся преимущественно именно на этом языке, так что портирование приложений, исходный код которых был написан на Java, постепенно теряет актуальность и применимость.

Стоит отметить, что подобные инструменты имеют такой недостаток, как отсутствие гарантий корректности выполняемых преобразований. Результат генерации кода, полученный с помощью нейросетевых моделей, не проходит строгую верификацию и может содержать логические несоответствия исходной программе. В частности, при переносе

Android-приложений это проявляется в неправильном сопоставлении жизненных циклов компонентов, различиях в обработке событий, а также в некорректном использовании системных API HarmonyOS.

Следует также дополнить, что теоретически возможным является и многошаговый сценарий портирования Android-приложений, при котором исходный код на языке Kotlin предварительно транслируется в Java, а затем уже обрабатывается инструментами, ориентированными на перенос Java-программ в ArkTS. Однако данный подход не устраняет фундаментальных недостатков вышеописанного подхода. Преобразование Kotlin-кода в Java неизбежно приводит к потере информации о специфических конструкциях исходного языка, усложняет восстановление семантики программы и затрудняет анализ архитектурных решений, заложенных разработчиком. В результате на следующем этапе портирования в ArkTS данные ограничения сохраняются и дополняются неопределённостью, характерной для инструментов, основанных на машинном обучении, что в совокупности снижает предсказуемость и воспроизводимость результата.

8. Заключение

Таким образом, был разработан подход, который позволяет автоматизировать процесс преобразования программ с языка Kotlin в язык ArkTS, применяемый в экосистеме HarmonyOS NEXT. В отличие от прямых конвертеров, ориентированных только на синтаксическую совместимость, данный инструмент выполняет семантический анализ, основанный на формальных спецификациях библиотек и проверке эквивалентности поведения программных конструкций.

Ключевой особенностью решения является использование формальных спецификаций на языке LibSL, которые позволяют описывать поведение библиотечных функций независимо от их реализации. В сочетании с использованием SMT-решателей это обеспечивает возможность автоматической проверки эквивалентности функций исходной и целевой платформ. Такой подход делает возможным корректное сопоставление даже в тех случаях, когда прямые аналоги между библиотеками Kotlin и ArkTS отсутствуют или имеют отличия в сигнатурах и семантике.

Разработанный подход демонстрирует, что формальные методы и языки формальной спецификации могут быть эффективно применены для задач трансляции между языками программирования, обеспечивая гарантированное сохранение семантики и повышение надежности результатов портирования.

Предложенный подход реализован в прототипе инструмента портирования, который показал применимость предложенного подхода, успешно портировав логику и библиотечное окружение несколько небольших приложений.

Развитием разработанного подхода является интеграция в него методов портирования пользовательских интерфейсов, а также формирование специальной тестовой инфраструктуры, позволяющей организовывать дополнительные проверки корректности портирования приложений из одной экосистемы в другую.

Список литературы / References

[1]. Алексюк, А.О., Ицыксон, В.М. Семантически-ориентированная миграция Java-программ: опыт практического применения. Modeling and Analysis of Information Systems, 2017, 24, 677-690.
[2]. Перцев П.В. Использование формальных спецификаций для автоматизации портирования программ: выпускная квалификационная работа (магистр) / Перцев П.В.; науч. рук. Ицыксон В.М. Санкт-Петербург: Университет ИТМО, 2023. 2025 с.
[3]. JetBrains, Kotlin Analysis API, Available at: https://github.com/Kotlin/analysis-api, accessed 01.11.2025.
[4]. Huawei, Learning ArkTS. Available at: https://developer.huawei.com/consumer/en/doc/harmonyos-guides/learning-arkts, accessed 01.11.2025.
[5]. Itsykson V.M. LibSL: Language for Specification of Software Libraries, 2018.

[6]. de Moura L.M., Bjørner N.S. Satisfiability modulo theories. Communications of the ACM, 2011, vol. 54, no. 9, pp. 69-77. DOI: 10.1145/1995376.1995394.
[7]. Petrov M., Gagarski K., Belyaev M., Itsykson V. Using a Bounded Model Checker for Test Generation: How to Kill Two Birds with One SMT-solver. Modeling and Analysis of Information Systems, 2014, vol. 21(6), pp. 83-93 (in Russian). DOI: 10.18255/1818-1015-2014-6-83-93.
[8]. JetBrains, Kotlin/JS Compiler. Available at: https://kotlinlang.org/docs/js-overview.html, accessed 01.11.2025.
[9]. Gong L., Wang C., Cui D., Huang Y., Wei M. UITrans: Seamless UI Translation from Android to HarmonyOS. Proceedings of the 16th International Conference on Internetware, 2024.
[10]. Liu R., Lin Y., Hu Y., Zhang Z., Gao X. LLM-Based Java Concurrent Program to ArkTS Converter. 2024 39th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2024, pp. 2403-2406.

Информация об авторах / Information about authors

Иван Александрович МИЛЕНИН – студент Института прикладных компьютерных наук Университета ИТМО. Сфера научных интересов: языки программирования, трансляция и транспиляция исходного кода, формальные спецификации, автоматизация миграции программных систем.

Ivan Aleksandrovich MILENIN – student at the Institute of Applied Computer Science, ITMO University. Research interests: programming languages, source code translation and transpilation, formal specifications, automation of software system migration.

Владимир Михайлович ИЦЫКСОН – кандидат технических наук, доцент института прикладных компьютерных наук ИТМО. Сфера научных интересов: статический и динамический анализ программ, верификация программного обеспечения, методы обнаружения дефектов в исходном коде, методы автоматизации тестирования программ.

Vladimir Mikhailovich ITSYKSON – Cand. Sci. (Tech.), associate professor at the Institute of Applied Computer Science ITMO. Research interests: static and dynamic software analysis, software verification, methods for detecting defects in source code, methods for automating software testing.