**УДК 004.4’23**

# Поддержка вызовов функций и передачи объектов между реализациями языков со статическими и динамическими типизациями в среде выполения на платформе Ark

**Прокопенко К.Д.** (Университет ИТМО)
**Научный руководитель – Соломенников Д.И.**

**Введение.** Существуют различные виртуальные машины для приложений, такие как JVM и .NET, или же Ark в случае данной работы, крупные компании предпочитают свои собственные разработки. Для реализации графических интерфейсов самым популярными языками является те, что удовлетворяют открытому стандарту ECMAScript, благодаря их распространенности в браузерах и упрощениям, которые дает динамическая типизация. Благодаря этому, компаниям проще нанимать разработчиков на таких языках, нежели искать специалистов на статически типизированном. Однако, продвинутая логика приложения, как и нижележащие графические библиотеки зачастую реализуются именно на статически типизированных языках, поскольку это повышает как стабильность, так и скорость работы, из-за чего и возникает необходимость во взаимодействии этих языков.

**Основная часть.** В рамках данной работы были рассмотрены такие решения, как LuaJIT, nashorn, JNI и другие. В результате сравнения их преимуществ, недостатков и ограничений были выделены следующие требования:

1) Необходимо расширить типовую систему, введя новый тип “инородных” объектов. Для этого в статически типизированном языке необходимо добавить закрытый тип, от которого, с точки зрения типовой системы будет наследоваться корневой тип (“Object”) и тип, представляющий объекты из динамически типизированного языка. Таким образом, можно будет построить типо-безопасный интерфейс, который может работать как с известным типом объекта, так и с произвольным. Со стороны динамического языка, т.к. реализация использует “NaN boxing”, необходимо ввести новый тэг, который затем корректно обработать в интерпретаторе и JIT компиляторе. Доступ к статическим полям и методам можно осуществить аналогичным образом. Во многом аналогичный подход реализует LuaJIT для структур из языка C [1] [2].

2) Управляемым программам необходимо некоторое состояние системы, которое выгодно хранить в локальной переменной потока в случае интерпретации, и в регистре в случае скомпилированного управляемого кода. Можно заметить, что оно отличается для различных языков, из-за чего при переключении языкового контекста необходимо подменять это значение в следующих местах: в нативном коде при переключении языка, в мосту между скомпилированным кодом и интерпретатором или средой выполнения

3) В интерпретаторе необходимо расширить сборщик профиля программы так, чтобы он не только сохранял динамические классы, но и конкретные поля и методы для инородных объектов. Для разрешения перегрузок может быть введен новый профиль, который будет сохранять типы аргументов, что, однако, замедлит выполнение программ, не использующих межъязыковое взаимодействие.

4) В JIT компиляции, помимо непосредственной генерации корректного доступа к полям и защиты для “деоптимизации”, необходимо добавить проход оптимизации, который будет заменять проверку принадлежности классу на сравнение с нулевым указателем, если экземпляр был получен из статически типизированного поля или метода. Помимо этого, семантика вызова метода отличается: в динамических языках нотация вызова через точку раскрывается не в вызов конкретного метода, а в получение свойства объекта с заданным именем, и затем применение его как функции, с установленной ссылкой “this” на корректное значение. Из-за этого необходимо не только подменить вызов на корректный, но и сохранить значение этого свойства на случай деоптимизации при вычислении аргументов

5) С точки зрения сборщика мусора, чтобы избежать сравнения на каждое посещение объекта, необходимо добавить новый режим, позволяющий работать как со статическими,

так и с динамическими объектами, путем обобщения двух уже существующих реализаций. В дополнение к этому, у реализаций может различаться расположение данных на стеке, что может помешать сборщику мусора его сканировать. Для решения этой проблемы, в момент переключения языков, должен создаваться новый временный кадр стека, обнаружив который, обработчик стека будет знать, что необходимо изменить текущее состояние языка [3]

**Выводы.** Проведен анализ существующих решений в области взаимодействия реализаций языков программирования, спроектировано композитное решение для платформы Ark, существенная часть которого уже реализована.

**Список использованных источников**:


1. LuaJIT FFI Library. — Текст : электронный // http://luajit.org : [сайт]. — URL: http://luajit.org/ext_ffi.html (дата обращения: 20.02.2023).
2. What I learned from LuaJIT. — Текст : электронный // https://mrale.ph : [сайт]. — URL: https://mrale.ph/talks/vmss16 (дата обращения: 20.02.2023).
3. Interaction of compiled code and the runtime on the Ark platform. — Текст : электронный // https://gitee.com : [сайт]. — URL: https://gitee.com/openharmony-sig/arkcompiler_runtime_core/blob/master/docs/runtime-compiled_code-interaction.md (дата обращения: 20.02.2023).