УНИВЕРСИТЕТ ИТМО
НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
май–июнь 2020 Том 20 № 3 ISSN 2226-1494 http://ntv.itmo.ru/
SCIENTIFIC AND TECHNICAL JOURNAL OF INFORMATION TECHNOLOGIES, MECHANICS AND OPTICS
May–June 2020 Vol. 20 No 3 ISSN 2226-1494 http://ntv.itmo.ru/en/
НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК УНИВЕРСИТЕТ ИТМО ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

УДК 004.45 doi: 10.17586/2226-1494-2020-20-3-410-417

МЕТОД ОБЕСПЕЧЕНИЯ ПЕРЕНОСИМОСТИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА ОСНОВЕ ПЕРЕНАЦЕЛИВАЕМОЙ СРЕДЫ ВЫПОЛНЕНИЯ ПРОГРАММ

И.П. Логинов[a], А.М. Дергачев[a], Е.А. Павловский[b]

[a] Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация
[b] Петербургский государственный университет путей сообщения Императора Александра I, Санкт-Петербург, 190031, Российская Федерация
Адрес для переписки: dam600@gmail.com

Информация о статье
Поступила в редакцию 18.04.20, принята к печати 20.05.20
Язык статьи — русский

Ссылка для цитирования: Логинов И.П., Дергачев А.М., Павловский Е.А. Метод обеспечения переносимости программного обеспечения на основе перенацеливаемой среды выполнения программ // Научно-технический вестник информационных технологий, механики и оптики. 2020. Т. 20. № 3. С. 410–417. doi: 10.17586/2226-1494-2020-20-3-410-417

Аннотация

Предмет исследования. Исследованы подходы к разработке переносимого программного обеспечения на уровне бинарного и исходного программного кода, а также факторы, влияющие на переносимость, такие как совместимость целевых платформ на уровне программных и бинарных интерфейсов приложений, стандартизация языков программирования, программная архитектура, функциональные возможности программных целевых платформ и наличие инструментального программного обеспечения. Рассмотрены современные подходы к обеспечению переносимости на основе виртуальных сред выполнения программ для языка Java и платформы .NET. **Метод.** Предложен метод обеспечения переносимости программного обеспечения на основе двухэтапной компиляции и применения языка описания архитектур для настройки транслятора, что позволяет решать задачу переносимости программного обеспечения на уровне среды выполнения программ, а также новый подход к реализации самонастраивающейся среды выполнения, параметрами конфигурации которой являются архитектурно-зависимые метаданные — описания целевых платформ. Для получения бинарного образа среды выполнения для заданной целевой платформы не требуется наличие ее исходного кода. Генерация образа выполняется на основе метаданных, входящих в состав существующего образа, который используется как утилита-построитель. **Основные результаты.** Определены требования к реализации среды выполнения программ, а также ее архитектура на уровне функциональных компонентов. Предложен новый подход к реализации среды выполнения, позволяющий обеспечить переносимость без перекомпиляции исходного кода как пользовательских приложений, так и самой среды выполнения. Разработан сценарий использования среды выполнения для генерации ее бинарного образа, нацеленного на заданную платформу. **Практическая значимость.** Обеспечение бинарной переносимости среды выполнения программ позволит снизить трудозатраты на реализацию кроссплатформенных приложений.

Ключевые слова
переносимость, кроссплатформенное программное обеспечение, среда выполнения, язык описания архитектур, двухэтапная компиляция

Благодарности
Работа выполнена в рамках проекта «Разработка методологической и технической основы для создания средства построения компиляторов для устройств Интернета вещей», поддержанного ФГБУ «Фонд содействия развитию малых форм предприятий в научно-технической сфере».

doi: 10.17586/2226-1494-2020-20-3-410-417

SOFTWARE PORTABILITY BASED ON RETARGETABLE RUNTIME ENVIRONMENT

I.P. Loginov[a], A.M. Dergachev[a], E.A. Pavlovskiy[b]

[a] ITMO University, Saint Petersburg, 197101, Russian Federation
[b] Emperor Alexander I[st] Petersburg State Transport University, Saint Petersburg, 190031, Russian Federation
Corresponding author: dam600@gmail.com

Article info
Received 18.04.20, accepted 20.05.20
Article in Russian

For citation: Loginov I.P., Dergachev A.M., Pavlovskiy E.A. Software portability based on retargetable runtime environment. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2020, vol. 20, no. 3, pp. 410–417 (in Russian). doi: 10.17586/2226-1494-2020-20-3-410-417

Abstract

Subject of Research. The paper presents research of approaches to portable software development at the level of binary and source code. We study such factors affecting portability as compatibility of target platforms at the level of software and binary application interfaces, standardization of programming languages, software architecture, the functionality of software target platforms and software tools. Modern approaches for software portability based on virtual runtimes for Java and the .NET platform are considered. **Method.** A method is proposed for software portability based on two-stage compilation and an architecture description language application for translator configuration. The method gives the possibility to solve the software portability problem at the level of program execution environment. We also present a new approach to self-tuning runtime implementation with such configuration parameters as architecture-dependent metadata — descriptions of target platforms. To generate a binary image of the runtime environment for a target platform, its source code is not required. Image generation is performed based on metadata that is part of an existing image used as a builder utility. **Main Results.** Implementation requirements for the program execution environment and its architecture at the level of functional components are determined. The novel approach for the runtime implementation is proposed which ensures portability without recompilation from the source code of both user applications and the runtime environment. A script for the runtime environment application has been developed to generate its binary image aimed at a target platform. **Practical Relevance.** Binary portability of the program execution environment allows reducing labor costs for cross-platform applications.

Keywords
portability, cross-platform software, execution environment, architecture description language, two-stage compilation

Acknowledgements
The study was performed as part of the project “Development of methodological and technical basis for creating compiler building tools for IoT devices” and supported by the grant from the “Fund for Development Promotion of Enterprise Small Forms in Scientific and Technical Sphere”.

Введение

Кроссплатформенность является важным атрибутом качества программного обеспечения (ПО), определяющим степень переносимости программных приложений между существующими и вновь создаваемыми программно-аппаратными платформами. Это влияет на востребованность ПО как среди пользователей, так и среди разработчиков, и в итоге – на его конкурентоспособность. Обеспечение кроссплатформенности ПО является важной задачей, решение которой связано с существенными затратами времени, финансов и человеческих ресурсов.

В зависимости от того, на каком этапе разработки к приложению предъявляется требование кроссплатформенности, существуют два сценария обеспечения переносимости:

1) при определении требований к новому приложению кроссплатформенность может быть обеспечена за счет использования инструментального ПО, языков программирования и библиотек, поддерживающих несколько платформ;
2) при необходимости обеспечить переносимость существующего приложения задача решается методом портирования — выполняется адаптация исходного кода для поддержки новой целевой платформы [1].

В случае абсолютной непереносимости исходного кода программы либо нецелесообразности его адаптации, кроссплатформенность может быть обеспечена за счет повторной разработки. При отсутствии исходного кода приложения, например, для устаревшего и проприетарного ПО, единственным возможным способом переноса на другие целевые платформы является использование средств виртуализации, что связано с ограничениями по возможности управления вычислительными ресурсами, производительности, совместимости аппаратуры [2, 3].

В настоящий момент существует множество методов и подходов для решения частных задач переносимости. Так, для обеспечения кроссплатформенности приложений на базе операционной системы Android предлагается использовать Google Web Toolkit, позволяющий создавать платформенно-независимые веб-приложения [4]. В работе [5] предлагается метод портирования, при котором специфичный для платформы код приложения изолируется и может быть преобразован для новой платформы в полуавтоматическом режиме при наличии подходящих API (Application Program Interface). Работы [6, 7] посвящены технологии разработки кроссплатформенного ПО, а [8] рассматривает основные приемы, характерные для портирования крупных программных продуктов. В [9] предлагается схема реструктуризации мобильных приложений, призванная в дальнейшем подготовить код к частичной автоматизации процесса портирования. Однако все перечисленные подходы обеспечения переносимости являются частными по отношению к отдельным типам приложений, например, к мобильным или веб-приложениям, либо ориентированы на отдельные спецификации программных интерфейсов приложений — API.

Реализация кроссплатформенного ПО или отдельных его компонентов позволяет обеспечить повторное использование программного кода [10], снизить затраты на тестирование [11] и дальнейшее сопровождение, поэтому создание метода реализации кроссплатформенного ПО является актуальной задачей, позволяющей

снизить издержки процесса разработки. Решением проблемы может стать разработка самонастраивающейся среды выполнения, обеспечивающей переносимость ПО без адаптации исходного кода как программного приложения, так и лежащей в его основе программной инфраструктуры, и инструментального ПО.

Терминология

Переносимость — атрибут качества ПО [12], который характеризует возможность использования приложения более чем на одной целевой платформе. Под целевой платформой (также применяется термин «целевое окружение») подразумевается программная или аппаратная платформа, на которой планируется использовать приложение. Целевые платформы могут быть представлены на аппаратном и программном уровнях. Отличия между целевыми платформами на аппаратном уровне заключаются в наборах команд, моделях памяти, режимах адресации, и других архитектурных особенностях. На программном уровне разница между целевыми платформами представляется в программных интерфейсах приложений.

Выделяют переносимость двух видов:

1) бинарную;
2) уровня исходного кода [13].

Бинарная переносимость предполагает возможность запуска приложения в виде исполняемого модуля на нескольких платформах. Бинарная переносимость приложения ограничивается набором очень схожих целевых платформ, совместимых на уровне API и двоичных (бинарных) интерфейсов приложений — ABI (Application Binary Interface).

Переносимость приложения на уровне исходного кода обеспечивается за счет возможности его компиляции для заданного целевого окружения без существенных затрат на адаптацию. Под существенными затратами на адаптацию следует понимать объем работы, сопоставимый с повторной разработкой приложения для заданной целевой платформы. Показательным примером является семейство операционных систем (ОС) UNIX. После того, как системное ядро было реализовано на языке Си без использования платформенно-зависимого языка ассемблера, ОС UNIX была адаптирована для множества аппаратных архитектур. Однако адаптации также способствовала модульная архитектура ОС UNIX [14], позволяющая вносить изменения строго в платформенно-зависимые компоненты, не затрагивая других.

Факторы, влияющие на переносимость

На переносимость ПО влияет множество факторов. Далее приводится их перечень, основанный на работе [15] и дополненный особенностями, связанными с созданием средств разработки ПО:

— стандартизация языков программирования;
— программные архитектурные решения;
— функциональные возможности программных окружений;
— инструментальное ПО.

Следование стандартам при разработке ПО позволяет обеспечить единообразное поведение программ, а также гарантировать совместимость при взаимодействии приложений или их компонентов. Однако при разработке ПО стандарты не всегда соблюдаются строго, что ограничивает переносимость как на уровне исходного кода, так и в бинарном формате. Также многие из распространенных языков программирования не стандартизованы организациями ISO, ECMA или ANSI, являясь в то же время стандартизованными де-факто. К таким языкам относятся Python, PHP, Objective-C, Elixir, Clojure, Erlang, Kotlin, и другие. Проблема таких языков заключается в том, что их реализации основаны на исследовании документации и поведения оригинальной версии языка программирования. Это негативно сказывается на разработке переносимых приложений, так как не позволяет гарантировать одинакового поведения программ для каждой реализации языка.

Возможности по адаптации исходного кода приложений для поддержки новых целевых платформ напрямую зависят от архитектуры приложения, так как она влияет на эффективность и сложность процесса разработки и сопровождения. Так, например, модульная архитектура приложений позволяет сделать процесс адаптации кода приложения эффективнее [16]. Принятие технических решений на уровне архитектуры оказывает влияние на весь процесс разработки — кодирование ПО, отладку отдельных компонентов и системы в целом.

На переносимость ПО влияют особенности программных окружений, поддержку которых требуется реализовать. К ним относятся программные интерфейсы приложений, предоставляемые разработчикам (Windows API, системные вызовы UNIX), функциональность стандартных библиотек языков программирования, сред выполнения программ (например, библиотеки классов .NET Framework и JDK). Как правило, API различных семейств ОС предоставляют схожую функциональность, но исключают возможность обеспечения бинарной переносимости, вследствие чего требуется адаптация исходного кода. Также важна совместимость на уровне двоичных интерфейсов (ABI), которые определяют механику вызова подпрограмм (соглашения о вызовах) и использования регистров процессора (например, с какой целью используются определенные регистры). Отсутствие совместимости на уровне ABI не позволит выполнить перенос приложения как минимум без перекомпиляции, а как максимум — без адаптации исходного кода.

Для того чтобы получить приложение для заданной целевой платформы, необходимо инструментальное ПО, поддерживающее эту платформу. Как минимум для используемого в разработке приложения языка программирования должен быть реализован компилятор или интерпретатор. Также процесс разработки должен быть поддержан и другим инструментарием — средствами отладки, диагностики и тестирования.

В настоящее время создано множество компиляторов, отладчиков и других средств, позволяющих работать с несколькими целевыми платформами. Так, например, семейство компиляторов GCC или инфра-

структура для разработки компиляторов LLVM поддерживают множество целевых платформ. В то же время реализация поддержки платформы, которая ранее не поддерживалась, связана с существенными трудозатратами. Данная проблема характерна для области разработки ПО встраиваемых вычислительных систем на основе конфигурируемых, проблемно-ориентированных процессоров. Подробнее проблемы разработки инструментального ПО рассмотрены в работе [17].

Виртуальные среды выполнения программ

В предыдущем разделе были перечислены факторы, оказывающие влияние на переносимость ПО преимущественно на уровне исходного кода. Необходимо минимизировать их влияние на переносимость. Одним из подходов к обеспечению переносимости является применение виртуальных сред выполнения программ, таких как Java Virtual Machine (JVM), Common Language Runtime (CLR). Виртуальная среда выполнения программы — система, реализующая виртуальную вычислительную машину с заданной моделью памяти, набором инструкций, системой типов, а также набором сервисов и библиотек, необходимых разработчикам. В основе таких сред выполнения лежит двухэтапная модель компиляции, согласно которой преобразование программы выполняется из кода на исходном языке в промежуточное представление (байт-код), а лишь затем среда выполнения генерирует машинный код. Таким образом, бинарный модуль для виртуальной машины может быть выполнен на любой платформе, для которой существует реализация данной виртуальной машины. Благодаря этому JVM и CLR стали применяться для решения задачи переносимости.

Применение термина «переносимость» для сред выполнения программ следует рассматривать как в контексте кода на исходном языке программирования, так и в контексте реализации самой среды выполнения. Реализация поддержки целевых платформ находится на уровне среды выполнения. Таким образом, если в коде программы на исходном языке не используется платформенно-специфическая функциональность, она является переносимой по умолчанию. Однако программа на исходном языке является переносимой в той степени, в которой поддерживаются целевые платформы средой выполнения. Исследование исходного кода сред выполнения JVM, CLR, TinyCLR показало, что такая поддержка реализуется статически, и для обеспечения их переносимости требуется адаптация исходного кода. Далее рассматривается предлагаемое решение — переносимая среда выполнения.

Предлагаемый метод обеспечения переносимости программного обеспечения

Проблему переносимости ПО можно рассмотреть на различных уровнях абстрагирования от целевой платформы:

1) аппаратные возможности, предоставляемые целевой платформой, такие как процессоры, средства ввода-вывода, системы хранения данных;
2) архитектура процессора, определяющая набор команд, множество регистров и режимов адресации;
3) API и ABI, реализованные на уровне ОС;
4) функциональность, предоставляемая библиотеками и системными вызовами ОС, которые могут быть кроссплатформенными.

Частично эти уровни абстракции реализованы в виртуальных машинах JVM и CLR, опыт использования которых показал, что виртуальные среды выполнения программ являются наиболее подходящим средством обеспечения переносимости отдельных программных приложений. В то же время остается нерешенной проблема переносимости самой виртуальной среды выполнения. Для обеспечения переносимости среды выполнения необходимо средство, предоставляющее функциональность на всех перечисленных выше уровнях абстракции.

Предлагаемый метод обеспечения переносимости ПО основан на использовании встроенного в среду выполнения перенацеливаемого компилятора — компилятора, организация исходного кода которого ориентирована на адаптацию с целью поддержки новых платформ. Получаем так называемую перенацеливаемую среду выполнения, настраиваемую на заданную целевую архитектуру. В отличие от существующих реализаций виртуальных сред выполнения программ, а также конфигурируемых компиляторов, предлагаемое решение является параметризуемым и не требует изменения исходного кода для реализации поддержки целевой платформы с последующей перекомпиляцией. Встроенный в состав среды выполнения перенацеливаемый компилятор реализует поддержку целевых платформ за счет применения их описаний при генерации кода, которые разрабатываются на предметно-ориентированном языке ADL (Architecture Description Language) [18].

Описание архитектуры на языке ADL существенно компактнее, чем программная реализация поддержки целевой платформы в коде среды выполнения. В табл. 1 и 2 приводится сравнение объемов реализации JIT-компиляторов, реализующих поддержку генерации кода для архитектуры ARM и x86 в разных средах выполнения. Столбец ADL содержит количество строк кода на языке ADL, в остальных столбцах — значения для языка C++. Описания архитектур на языке ADL на

Таблица 1. Объем реализаций генераторов кода для архитектуры ARM

Единицы измерения	TinyCLR	CLR	ADL
Тысяч строк кода, физических	6,7	6,1	2,1
Тысяч строк кода, логических	4,7	3,1	1,5

Таблица 2. Объем реализаций генераторов кода для архитектуры x86

Единицы измерения	HotSpot	CLR	ADL
Тысяч строк кода, физических	28,1	10,3	6,4
Тысяч строк кода, логических	17,4	4,5	3,4

25–65 % компактнее, чем реализация JIT-компиляторов в средах выполнения TinyCLR и CLR, а также в версии JVM, получившей название HotSpot.

Описание целевой платформы, в зависимости от уровня абстракции платформы, может включать в себя описания отдельных аспектов целевой платформы:
— структуры и поведения аппаратуры;
— архитектуры набора команд;
— отображения функциональности ОС на функциональность среды выполнения;
— отображения функциональности необходимых высокоуровневых библиотек на функциональность среды выполнения или отдельного приложения.

Выбор уровня абстракции позволяет достигнуть переносимости среды выполнения для заданной целевой платформы. При этом более избыточное описание позволит генерировать более эффективный машинно-ориентированный код, но его создание не является необходимым для решения задачи переносимости.

Схематически процесс преобразования программы из исходного кода в код на целевой платформе с использованием среды выполнения показан на рис. 1. Среда выполнения в качестве входных данных принимает не только код программы на промежуточном языке, но и описание целевой платформы. Процедуры преобразования кода программы: A — из кода на языке программирования в промежуточное представление программы для среды выполнения; B — из промежуточного представления в машинно-ориентированный код для целевой платформы.

Основываясь на вышесказанном, предлагаемый метод обеспечения переносимости ПО на заданную целевую платформу можно описать следующей последовательностью шагов:
1) определить подмножество платформенно-специфической функциональности приложения и среды выполнения;
2) реализовать описание архитектуры целевой платформы на языке ADL;
3) выполнить конфигурирование среды выполнения данным описанием;
4) получить бинарный образ среды выполнения для заданной целевой платформы;
5) использовать бинарные образы приложений, поддерживаемые средой выполнения.

Платформенно-зависимые аспекты реализации среды выполнения

Разработанная на основе предложенного метода перенацеливаемая среда выполнения программ соответствует следующим требованиям.
1. Генерация кода основана на описании целевых платформ на языке описания архитектур. Статическая реализация платформенно-специфической функциональности не предусматривается, либо такая функциональность должна быть представлена в виде подключаемого модуля.
2. Предусматривается выбор между интерпретатором или компилятором для генерации кода.
3. Среда выполнения может генерировать бинарный образ самой себя без перекомпиляции из исходного кода.
4. Компоненты среды выполнения конфигурируемы и реализованы как модули.

Генерация кода на основе описаний целевых платформ на языке ADL является ключевым аспектом в реализации перенацеливаемой среды. Генератор кода учитывает такие аспекты как: архитектура памяти и регистров, набор поддерживаемых инструкций, порядок байтов, режимы адресации, особенности внутренних типов данных процессора, механизма обработки прерываний. Представленный в работе [19] язык ADL поддерживает всю необходимую функциональность и поэтому выбран для реализации механизма конфигурирования среды выполнения.

Выбор между интерпретатором и компилятором для генерации кода обусловлен, с одной стороны, простотой реализации интерпретатора в сравнении с компилятором, с другой — достаточной производительностью генерируемого кода для решения пользовательских задач.

Автоматическая генерация бинарного образа среды выполнения для заданной целевой платформы является ключевой возможностью по обеспечению переносимости, так как не требует наличия набора инструментального ПО, поддерживающего заданную целевую платформу, и исходного кода среды выполнения.

Реализация генерации бинарного образа возможна за счет наличия метаданных. Промежуточное представление программы (байт-код) сопровождается метаданными, которые предоставляют исчерпывающую

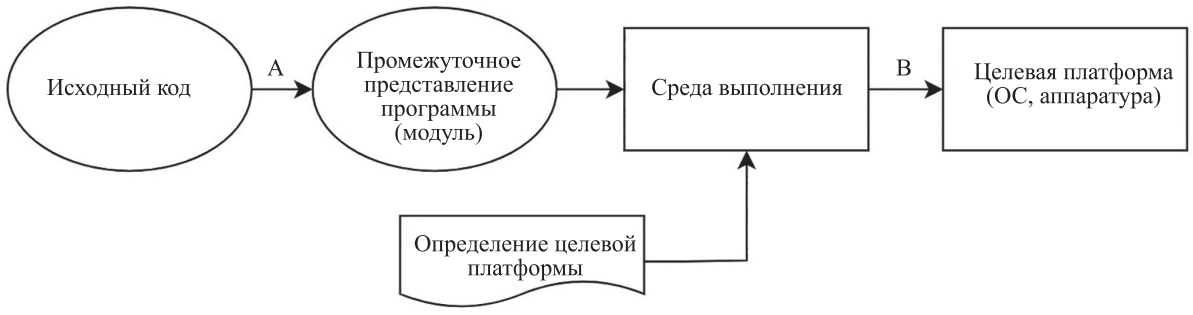

Рис. 1. Конфигурируемая среда выполнения

информацию обо всех типах данных и их элементах (методах, полях). Для реализации виртуальной машины выбрана модель метаданных, определенная стандартом ECMA-335, согласно которому реализуются виртуальные машины .NET Framework, .NET Core. Следование данному стандарту также позволяет обеспечить поддержку множества уже разработанных приложений. Конфигурируемые компоненты среды выполнения позволяют оптимизировать ее для использования на целевых платформах различной архитектуры и назначения. Так настройка менеджера памяти позволяет оптимизировать быстродействие в зависимости от архитектуры целевой платформы и предполагает возможности по использованию нескольких методов управления ресурсами. Конфигурирование системы типов предполагает создание ограничений по использованию примитивных типов данных согласно стандарту ECMA-335. На рис. 2 приводится набор компонентов среды выполнения программ и библиотеки классов. На данной схеме серым цветом выделена среда выполнения, а также подмножество библиотеки классов, необходимое для ее реализации. Компоненты библиотеки классов, которые могут быть непосредственно использованы разработчиками ПО для данной среды выполнения, отделены пунктирной линией от компонентов среды выполнения. Каждый компонент среды выполнения является конфигурируемым при помощи языка ADL.

Сценарий переноса среды выполнения

Для получения версии бинарного образа виртуальной среды выполнения программ, поддерживающей заданную целевую платформу, необходимо выполнить конфигурирование существующего бинарного образа набором параметров.

Для этого необходимо выполнить запуск среды выполнения с соответствующими параметрами, перечисленными в табл. 3.

Пример использования: *portable.exe -tdef armv8.pdsl -image pe.iff -mm cgc -codegen jit*

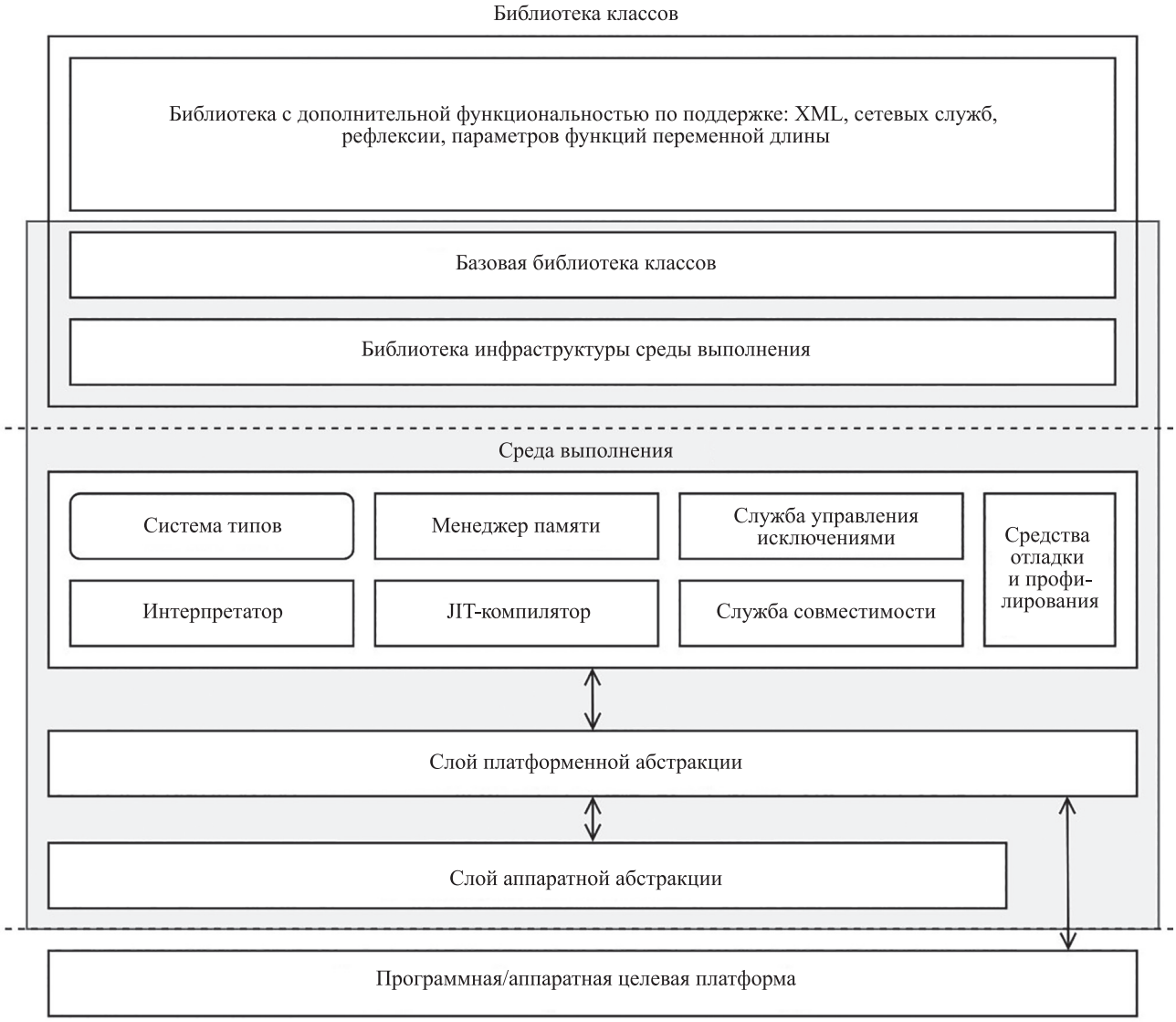

Рис. 2. Компоненты среды выполнения и библиотеки классов

Таблица 3. Параметры запуска среды выполнения

Параметр	Значение	Описание
-tdef	<path.pdsl>	Задает путь к файлу с описанием целевой платформы на языке описания архитектур
-mm	cgc, gc, sa, ssa	cgc — конкурентная сборка мусора; gc — сборка мусора без поддержки многопоточности; sa — автоматическое выделение и освобождение памяти на стеке; ssa — ручное выделение памяти на стеке и автоматическое освобождение
-codegen	-jit, -interp	Определяет способ генерации кода для целевой платформы: JIT-компиляция или интерпретация
-image	<path>	Путь к спецификации формата бинарного образа

В данном случае будет сгенерирован бинарный образ portable-armv8.exe, который нацелен для запуска на процессоре с архитектурой ARM, поддержкой конкурентной сборки мусора и JIT-компиляции. Формат образа pe.iff определяет то, что бинарный образ должен быть сгенерирован в формате PE (Portable Executable).

Теперь, если программа не использует специфических аппаратных возможностей, она может быть выполнена на целевой платформе без перекомпиляции.

Заключение

В работе предложен метод обеспечения переносимости программного обеспечения, основанный на описании целевой платформы на языке ADL, которое позволяет перенастраивать среду выполнения на заданную целевую платформу. Предложенный метод основан на идее реализации конфигурируемой среды выполнения, где под конфигурацией понимается передача среде выполнения в качестве параметров полного описания целевой платформы на специализированном формальном языке — ADL, что позволяет сократить объем платформенно-специфического программного кода как среды выполнения, так и входящего в ее состав перенацеливаемого компилятора на 25–65 % (в зависимости от сложности архитектуры и детализации ее описания) по сравнению с компиляторами, разработанными под конкретную архитектуру без применения языка ADL. Тестовая версия среды выполнения реализует базовый набор служб по управлению памятью и генерации кода, библиотеку классов. Полученное решение позволяет обеспечить бинарную переносимость как приложений, так и среды выполнения программ, и в дальнейшем может быть использовано для реализации единой автоматизированной системы обеспечения переносимости программного обеспечения на различные платформы без изменения исходного кода.

Литература

1. Hook B. Write Portable Code: An Introduction to Developing Software for Multiple Platforms. No Starch Press, 2005. 272 p.
2. Kozuch M.A., Kaminsky M., Ryan M.P. Migration without Virtualization // Proc. of HotOS'09: 12th Workshop on Hot Topics in Operating Systems. 2009.
3. Sahoo J., Mohapatra S., Lath R. Virtualization: A survey on concepts, taxonomy and associated security issues // Proc. 2nd International Conference on Computer and Network Technology (ICCNT 2010). 2010. P. 222–226. doi: 10.1109/ICCNT.2010.49
4. Klima P., Selinger S. Towards platform independence of mobile applications metamorphosing android applications for the web // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2013. V. 8112. Part 2. P. 442–449. doi: 10.1007/978-3-642-53862-9-56
5. Stehle T., Riebisch M. A porting method for coordinated multiplatform evolution // Journal of Software: Evolution and Process. 2019. V. 31. N 2. P. e2116. doi: 10.1002/smr.2116
6. Mooney J.D. Bringing Portability to the Software Process: Technical Report TR 97-1. Dept. of Statistics and Computer Science, West Virginia Univ., Morgantown, WV, 1997. 9 p.
7. Mooney J.D. Developing portable software // IFIP Advances in Information and Communication Technology. 2004. V. 157. P. 55–84. doi: 10.1007/1-4020-8159-6_3
8. Kagström S. Tools, techniques, and trade-offs when porting large software systems to new environments: Doctoral Dissertation. Blekinge Institute of Technology, 2008. 176 p.
9. Stehle T., Riebisch M. Establishing common architectures for porting mobile applications to new platforms // Softwaretechnik-Trends. 2015. V. 35. N 2.
10. Washizaki H., Yamamoto H., Fukazawa Y. A metrics suite for measuring reusability of software components // Proc. 5th

References

1. Hook B. *Write Portable Code: An Introduction to Developing Software for Multiple Platforms*. No Starch Press, 2005, 272 p.
2. Kozuch M.A., Kaminsky M., Ryan M.P. Migration without Virtualization. *Proc. of HotOS'09: 12th Workshop on Hot Topics in Operating Systems*, 2009.
3. Sahoo J., Mohapatra S., Lath R. Virtualization: A survey on concepts, taxonomy and associated security issues. *Proc. 2nd International Conference on Computer and Network Technology (ICCNT 2010)*, 2010, pp. 222–226. doi: 10.1109/ICCNT.2010.49
4. Klima P., Selinger S. Towards platform independence of mobile applications metamorphosing android applications for the web. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2013, vol. 8112, part 2, pp. 442–449. doi: 10.1007/978-3-642-53862-9-56
5. Stehle T., Riebisch M. A porting method for coordinated multiplatform evolution. *Journal of Software: Evolution and Process*, 2019, vol. 31, no. 2, pp. e2116. doi: 10.1002/smr.2116
6. Mooney J.D. *Bringing Portability to the Software Process*. Technical Report TR 97-1. Dept. of Statistics and Computer Science, West Virginia Univ., Morgantown, WV, 1997, 9 p.
7. Mooney J.D. Developing portable software. *IFIP Advances in Information and Communication Technology*, 2004, vol. 157, pp. 55–84. doi: 10.1007/1-4020-8159-6_3
8. Kagström S. *Tools, techniques, and trade-offs when porting large software systems to new environments*. Doctoral Dissertation. Blekinge Institute of Technology, 2008, 176 p.
9. Stehle T., Riebisch M. Establishing common architectures for porting mobile applications to new platforms. *Softwaretechnik-Trends*, 2015, vol. 35, no. 2.
10. Washizaki H., Yamamoto H., Fukazawa Y. A metrics suite for measuring reusability of software components. *Proc. 5th International*

International Workshop on Enterprise Networking and Computing in Healthcare Industry (IEEE Cat. No. 03EX717). 2004. P. 211–223. doi: 10.1109/METRIC.2003.1232469
11. Boehm B. Managing software productivity and reuse // Computer. 1999. V. 32. N 9. P. 111–113. doi: 10.1109/2.789755
12. Бурый А.С., Морин Е.В. Оценивание программных средств по множеству признаков // Известия высших учебных заведений. Приборостроение. 2019. Т. 62. № 10. С. 907–913. doi: 10.17586/0021-3454-2019-62-10-907-913
13. Mooney J.D. Issues in the specification and measurement of software portability // Poster session at the 15th International Conference on Software Engineering. 1993.
14. Raymond E.S. The Art of UNIX Programming. Addison-Wesley Professional, 2003. 560 p.
15. Tanenbaum A.S., Klint P., Bohm W. Guidelines for software portability // Software: Practice and Experience. 1978. V. 8. N 6. P. 681–698. doi: 10.1002/spe.4380080604
16. Mooney J.D. Portability and reusability: common issues and differences // Proc. of the 1995 ACM 23rd Annual Conference on Computer Science. 1995. P. 150–156. doi: 10.1145/259526.259550
17. Korenkov I., Loginov I., Doronin O., Sadyrin D., Dergachev A. Retargetable compiler design issues // Proc. 19th International Multidisciplinary Scientific GeoConference: SGEM. 2019. V. 19. N 2.1. P. 561–568. doi: 10.5593/sgem2019/2.1/S07.074
18. Qin W., Malik S. Architecture description languages for retargetable compilation // The Compiler Design Handbook: Optimizations and Machine Code Generation. 2002. P. 535–564. doi: 10.1201/9781420040579
19. Korenkov I., Loginov I., Dergachev A., Lazdin A. Declarative target architecture definition for data-driven development toolchain // Proc. 18th International Multidisciplinary Scientific GeoConference: SGEM: Surveying Geology & Mining Ecology Management. 2018. V. 18. N 2.1. P. 271–278. doi: 10.5593/sgem2018/2.1/S07.035

Workshop on Enterprise Networking and Computing in Healthcare Industry (IEEE Cat. No. 03EX717), 2004, pp. 211–223. doi: 10.1109/METRIC.2003.1232469
11. Boehm B. Managing software productivity and reuse. *Computer*, 1999, vol. 32, no. 9, pp. 111–113. doi: 10.1109/2.789755
12. Buryy A.S., Morin E.V. Software assessment by a set of indicators. *Journal of Instrument Engineering*, 2019, vol. 62, no. 10, pp. 907–913. (in Russian). doi: 10.17586/0021-3454-2019-62-10-907-913
13. Mooney J.D. Issues in the specification and measurement of software portability. *Poster session at the 15th International Conference on Software Engineering*. 1993.
14. Raymond E.S. The *Art of UNIX Programming*. Addison-Wesley Professional, 2003, 560 p.
15. Tanenbaum A.S., Klint P., Bohm W. Guidelines for software portability. *Software: Practice and Experience*, 1978, vol. 8, no. 6, pp. 681–698. doi: 10.1002/spe.4380080604
16. Mooney J.D. Portability and reusability: common issues and differences. *Proc. of the 1995 ACM 23rd Annual Conference on Computer Science*, 1995, pp. 150–156. doi: 10.1145/259526.259550
17. Korenkov I., Loginov I., Doronin O., Sadyrin D., Dergachev A. Retargetable *Compiler Design Issues. Proc. 19th International Multidisciplinary Scientific GeoConference: SGEM*, 2019, vol. 19, no. 2.1, pp. 561–568. doi: 10.5593/sgem2019/2.1/S07.074
18. Qin W., Malik S. Architecture description languages for retargetable compilation. *The Compiler Design Handbook: Optimizations and Machine Code Generation*, 2002, pp. 535–564. doi: 10.1201/9781420040579
19. Korenkov I., Loginov I., Dergachev A., Lazdin A. Declarative target architecture definition for data-driven development toolchain. *Proc. 18th International Multidisciplinary Scientific GeoConference: SGEM: Surveying Geology & Mining Ecology Management*, 2018, vol. 18, no. 2.1, pp. 271–278. doi: 10.5593/sgem2018/2.1/S07.035

Авторы

Логинов Иван Павлович — ассистент, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, ORCID ID: 0000-0002-6254-6098, ivan.p.loginov@gmail.com

Дергачев Андрей Михайлович — кандидат технических наук, доцент, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, Scopus ID: 57205166082, ORCID ID: 0000-0002-1754-7120, dam600@gmail.com

Павловский Евгений Алексеевич — кандидат технических наук, доцент, Петербургский государственный университет путей сообщения Императора Александра I, Санкт-Петербург, 190031, Российская Федерация, ORCID ID: 0000-0001-5411-5581, evgeny@pavlovskiy.spb.ru

Authors

Ivan P. Loginov — Assistant, ITMO University, Saint Petersburg, 197101, Russian Federation, ORCID ID: 0000-0002-6254-6098, ivan.p.loginov@gmail.com

Andrey M. Dergachev — PhD, Associate Professor, ITMO University, Saint Petersburg, 197101, Russian Federation, Scopus ID: 57205166082, ORCID ID: 0000-0002-1754-7120, dam600@gmail.com

Evgeny A. Pavlovskiy — PhD, Associate Professor, Emperor Alexander Ist Petersburg State Transport University, Saint Petersburg, 190031, Russian Federation, ORCID ID: 0000-0001-5411-5581, evgeny@pavlovskiy.spb.ru