Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики

Магистерская диссертация

Моделирование использования памяти в системе разработки на базе JME

Работу выполнил:
студент
Сафронов Андрей Юрьевич

Научный руководитель:
с.н.с. лаборатории ОИТ факультета ВМК МГУ им. М.В.Ломоносова,
к.ф.-м.н. *Намиот Дмитрий Евгеньевич*

Москва
2012

Оглавление

1. Аннотация

В данной работе рассматривается проблематика управления памятью в системах разработки программного обеспечения на базе JME. Автором проведено исследование модели управления памятью, которая требуется для работы программы в среде разработки на базе JME. Среда разработки ориентирована на создание программного обеспечения для встраиваемых систем, что накладывает определенные ограничения на ресурсы, которые могут быть использованы программой в процессе работы. Модель управления памятью основана в большей степени на статическом анализе. При этом статический анализ, по сути, является вспомогательной системой сборки мусора в процессе исполнения.

2. Введение

2.1 Встраиваемые системы.

Встраиваемая система (embedded system) – это компьютерная система, спроектированная для специальных функций управления внутри большой системы, часто с жесткими условиями по временной задержке реакции на события. По сравнению со встраиваемыми системами, персональный компьютер спроектирован для решения широкого круга задач пользователей. Встраиваемая система проектируется для решения более узкого круга задач.

Встраиваемые системы управляют большим количеством устройств в сегодняшнем мире.

Сферы использования варьируются от портативных устройств, такие как цифровые часы и mp3 плееры, автомобильные системы, бытовая техника и т.д., до больших стационарных инсталляций, таких как контроллеры для управления производственным процессом на заводах и систем, контролирующих ядерные электростанции.

Встраиваемые системы содержат процессорные ядра, которые в большинстве представляют собой микроконтроллер или цифровой сигнальный процессор (DSP).

Классы ядер, используемых во ВС очень широк, начиная от 8,16-битных микроконтроллеров и заканчивая 32 битными микроконтроллерами типа ARM, MIPS, PowerPC с частотой десятки и сотни мегагерц, внешней RAM и FLASH памятью объемами в десятки и сотни мегабайт.

Так как встраиваемые системы предназначены для решения специализированных задач, то инженеры имеют возможность оптимизировать дизайн систем таким образом, чтобы снизить размер и стоимость продукта и увеличить надежность и производительность. С

этой целью очень широко применяются системы на кристалле (***System-on-a-Chip***) при проектировании устройств.

Типичная система на кристалле содержит в себе процессорное ядро, банк памяти RAM и FLASH, а так же различные периферийные устройства(аналоговые преобразователи, внешние интерфейсы USB, Ethernet, SPI). За счет высокой интегрированности устройств различного назначения, получается значительная экономия в размерах, энергопотреблении, а также стоимости проектирования конечных устройств. Благодаря этим характеристикам SoC пользуются большой популярностью, например, одна только компания MICROCHIP к 2011 году поставила заказчикам 10 миллиардную микросхему SoC семейства PIC, это больше чем число процессоров в персональных компьютерах.

2.2 Обзор средств разработки программного обеспечения для встраиваемых систем.

Не смотря на широкое распространение встраиваемых систем с ограниченными ресурсами, нельзя назвать широко распространенных средств разработки. Наиболее используемая практика – программирование на языке C,C++ и assembler.

Многие компании-разработчики микропроцессоров предлагают пользователям свои средства разработки.

Компания Microchip предлагает для своих продуктов среду разработки MPLAB IDE[7], включающую средства компиляции на базе gcc и средства отладки, графические, сетевые библиотеки и библиотеки вывода. Весь это инструментарий можно использовать в основном только с продуктами компании Microchip.

Компания Atmel производит очень популярные семейства систем на кристалле C51, AVR и так же предлагает свою среду разработки AVR

Studio[9]. Компания Texas Instruments поддерживает свой инструментарий — Code Composer Studio[10].

Такой подход к выпуску инструментов разработки программного обеспечения для встраиваемых систем со стороны компаний-производителей систем на кристалле имеет свои минусы. Зачастую компании-разработчики параллельно разрабатывают программные библиотеки для своих продуктов схожие по функциональности, но ограниченные по применению в рамках свое продукции. При таком подходе появляются определенные трудности, связанные с разработкой качественных графических библиотек, например.

Некоторые компании разработчики ПО, предлагают свои решения в области тех же графических библиотек, например компания Amulet Technologies является одним из известных разработчиков высокопроизводительных GUI решений для встраиваемых систем. Компания поставляет свою систему разработки GEMstudio GUI Design Software[11].

В мире десктоп-систем и мобильных систем с достаточным количеством ресурсов существуют распространенные способы решения таких проблем, например разработчики для создания кроссплатформенного программного обеспечения могут использовать Qt библиотеки или платформу разработки Java. К сожалению, такие средства разработки не применимы к большинству систем на кристалле в силу ограниченности вычислительных ресурсов. В системах на кристалле зачастую отсутствует ОС и размер оперативной памяти измеряется 10кб.

Кроме разработки программного инструментария для встраиваемых систем с нуля, существует подход к разработке инструментария, основанный на адаптации программного обеспечения для десктоп-систем и мобильных систем с большим количеством вычислительных

ресурсов. То есть, на инструментарий для разработки программного обеспечения, накладываются определенные ограничения, и этот инструментарий адаптируется к области встраиваемых систем.

Можно выделить для рассмотрения две технологии разработки программного обеспечения для систем уровня десктоп, которые в настоящее время адаптируются для встраиваемых систем. Это платформа .NET и платформа JEE/JME

Такая крупная компания как Microsoft не обошла своим вниманием рынок встраиваемых систем и представила свое решение для встраиваемых систем — адаптированную версию платформы .NET[1].

Платформа разработки .NET MicroFrameWork.

Для разработки встраиваемых систем, преимущества .NET Micro FrameWork, могут быть обобщены следующим образом:

- низкий уровень стоимости разработки, чем у традиционных встраиваемых платформ;
- более быстрый выход готового продукта на рынок;
- более низкая стоимость устройств managed platforms;
- меньший размер устройств managed platforms;
- более низкое энергопотребление managed platforms;
- нет ограничений по специфическому чипсету;
- важная часть Microsoft embedded strategy.

Следующие возможности обеспечивают снижение стоимости разработки и более быстрый выход готового продукта на рынок, чем у

традиционных встраиваемых платформ:

- управляемый код и все его преимущества;
- программирование на современном языке с помощью Visual C#;
- широкий выбор библиотеки базовых классов (подмножество обычной платформы .NET);
- возможность низкоуровневой работы с устройствами с помощью драйверов;
- интеграция с Visual Studio;
- большое сообщество .NET разработчиков;
- возможность прототипирования и отладки с помощью эмулятора.

Платформа .NET MFW позволяет разрабатывать приложения для встраиваемых систем с использованием языка C# и Visual Studio.

При использовании .NET MFW не требуется наличие операционной системы во встраиваемых устройствах, MFW может выполняться низкоуровневым способом, т. к. MFW содержит сервисы, обычно предоставляемые операционной системой:

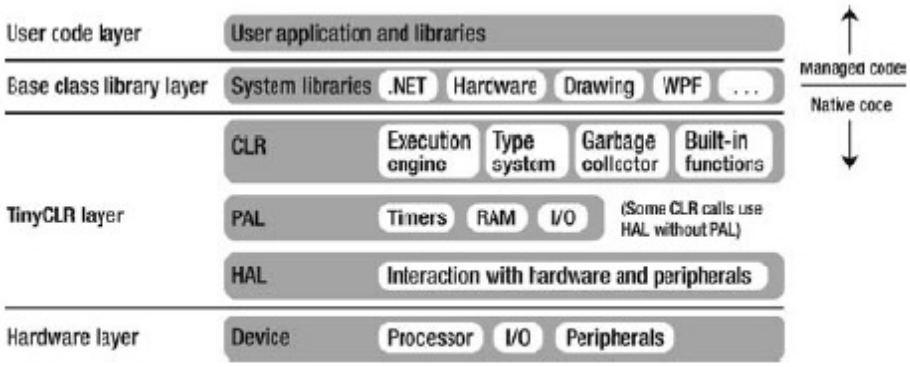
- загрузочный код;
- исполнение кода.

Архитектура .NET MicroFrameWork.

В последних версиях .NET MicroFrameWork разработчикам удалось снизить требования к минимальной конфигурации встраиваемой системы до 256kb Flash памяти и 64kb оперативной памяти,

разрядность процессора — 32 бит. Для исполнения кода, .NET MicroFrameWork использует интерпретатор. Указанные минимальные характеристики в виде разрядности процессора и интерпретатора ограничивают сферу применения .NET MicroFrameWork.

Ниже приведена иллюстрация из работы [1].



К платформе JEE/JME также наблюдается интерес со стороны производителей средств разработки для встраиваемых систем.

Виртуальная машина KVM

Компания Sun/Oracle предлагает свою технологию, для разработки ПО для встраиваемых систем. Предлагаемая технология, основана на платформе J2ME и виртуальной машине KVM [2].

Виртуальная машина K (KVM), являющаяся ключевым элементом архитектуры J2ME и представляет собой переносимую виртуальную машину, спроектированную заново для устройств с маленькой памятью, ограниченными ресурсами, и которые подключены к сетям, такие как сотовые телефоны и т.д. Эти устройства обычно включают в себя 16-битный или 32-битный процессор и с минимум оперативной памяти в районе 128 килобайт. Однако, KVM может быть гибко

развернута на широком классе устройств для различных применений и с широким диапазоном вариаций процессоров, размера памяти, характеристик устройств.

Общая идея дизайна KVM была в создании наименьшей возможной «полной» Java виртуальной машины, которая будет поддерживать все главные аспекты языка программирования Java, но может быть запущена в устройстве с ограниченными ресурсами с несколькими сотнями kB оперативной памяти.

Более подробно, KVM должна была удовлетворять требованиям:

- занимать мало места, от 40 до 80 kB;
- быть легко переносимой;
- быть модульной и конфигурируемой;
- быть «полной» и «быстрой» насколько это возможно.

Платформа J2ME вместе с KVM подходит для встраиваемых систем с объемом RAM больше 128к и достаточно высоким быстродействием CPU, поскольку Java байт-код интерпретируется виртуальной машиной. Но для большинства систем на кристалле, виртуальная машина KVM все же довольно велика по объему занимаемой памяти и недостаточно эффективна по быстродействию вследствие интерпретации.

Excelsior JET Embedded.

Компания Excelsior предлагает программное решение Excelsior JET Embedded[5], для использования Java во встраиваемых системах. Вместо медленной интерпретирующей байт-код виртуальной VM, Excelsior JET включает в себя оптимизирующий Ahead-Of-Time (AOT) компилятор. Java разработчики могут использовать его для

трансформации jar файлов приложения в оптимизированный исполняемый бинарный код целевой архитектуры. В результате, Java приложения исполняются на высокой скорости с момента запуска приложения. Компилятор AOT использует jar и class файлы на входе и создает исполняемый код на выходе, runtime так же включает в себя JIT компилятор для обработки классов, которые не были предварительно скомпилированы AOT.

Все конечно выглядит хорошо, тем не менее, для вычислительных мощностей embedded системы предъявляются достаточно высокие требования по ресурсам и быстродействию.

Требования к системе:

CPU	Intel Pentium II, 200 MHz и выше
RAM	5 MB минимум, рекомендуется 16+MB
ROM/FLASH	Минимум 15 MB
OS	Windows Linux: kernel 2.6.x, glibc 2.3+, NPTL 2.3+

Java платформа MicroEJ.

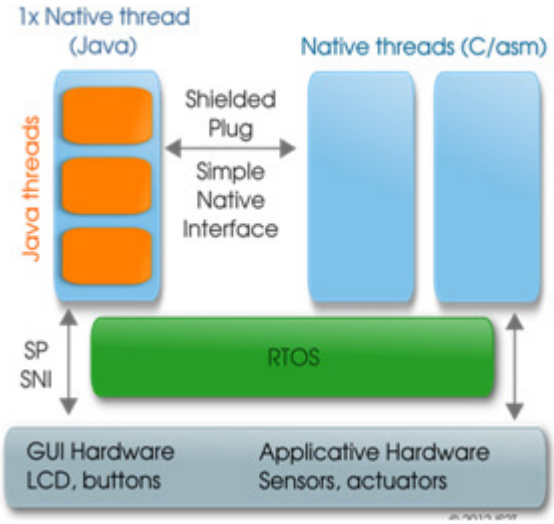
Компания IS2T предлагает свой подход к использованию Java для разработки приложений для встраиваемых систем [15].

Компанией разработана своя версия Java машины с учетом вычислительных возможностей встраиваемых систем. Виртуальная машина занимает в минимальном виде 28кб Flash памяти и 1кб оперативной памяти.

Есть варианты использования разработанной виртуальной машины как в составе с RTOS так и без нее.

Ниже представлен рисунок из статьи [26], показывающий

интеграцию разработанной JVM с RTOS.



Компания IS2T также предлагает свою реализацию библиотек, в том числе графических.

В данной платформе предусмотрен JNI интерфейс для написания низкоуровневых приложений.

В целом решение на базе Java для разработки программного обеспечения для встраиваемых систем от компании IS2T можно назвать интересным и заслуживающим внимание. Такой подход приносит некоторое новаторство в область встраиваемых систем. Компания за свой продукт также была отмечена различными наградами. Так, на конференции в Париже в 2012 году «5th Assises de l'Embarqué Conference», компания получила приз в категории «Embedded Technologies», что говорит об актуальности такого подхода.

Вместе с тем, данное решение обладает определенными недостатками. Прежде всего это наличие виртуальной машины, т. е. Java байт-код интерпретируется и выполняется существенно медленнее того же кода написанного на языке C. Второй недостаток, это ограниченный набор платформ, для которых сделана реализация виртуальной машины, в данный момент реализованы только версии для ARM и AVR32.

Исследовательские работы в области Java для встраиваемых систем

В академических кругах также наблюдается интерес к тематике использования Java для встраиваемых систем. Можно отметить два проекта в этой области.

Первый проект – это виртуальная машина Java для встраиваемых систем HVM (Hardware near Virtual Machine) [15]. В разработке данной среды принимают участие сотрудники VIA University College и Aalborg University. С результатами исследований по данному проекту можно ознакомиться в статьях [17][18][22]

Среда разработки HVM интегрирована с существующими методами разработки, использующие язык C. Для работы HVM в минимальной конфигурации требуется 20kb ROM и 500 байт RAM. Также проведена интеграция HVM с IDE Eclipse, сделана поддержка загрузки и обновлений Java приложений.

Ключевые особенности HVM:

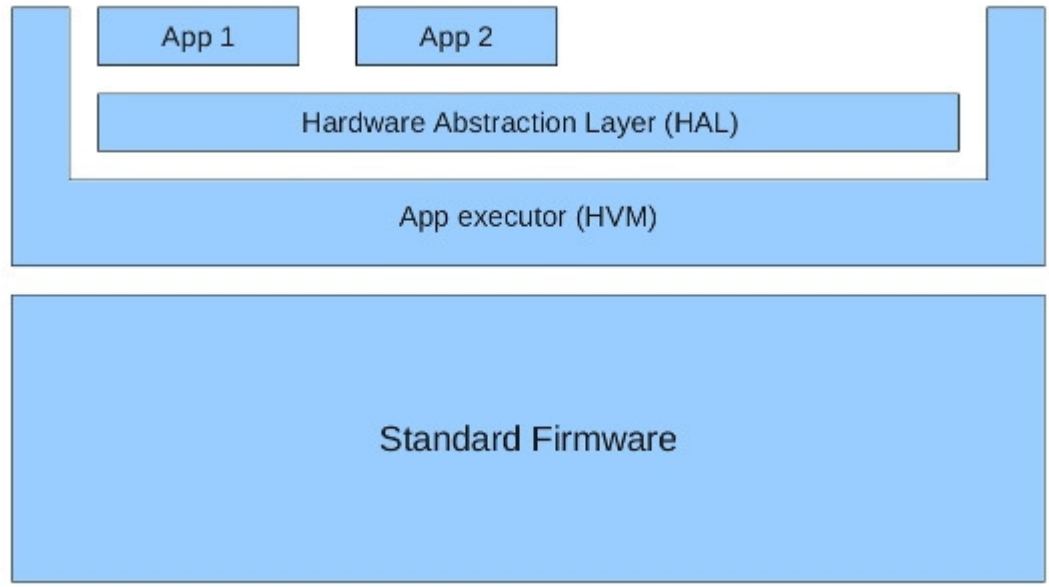
- вместе с приложением компонуются только те библиотеки, которые могут быть доступны во время выполнения;
- независимость от SDK. HVM работает со стандартными библиотеками Java из SDK Sun, так же могут быть использованы библиотеки из SDK других производителей;
- не требует для работы операционной системы;
- простая процедура сборки приложений;
- HVM поддерживает работу с низкоуровневыми (аппаратными) объектами и обработку прерываний низкого уровня;
- минимизация использования памяти RAM и ROM.

Интерпретатор занимает 30kb ROM и размер Java кучи по умолчанию 4kb;

- комбинированный способ выполнения. Некоторое подмножество методов может быть скомпилировано с помощью Ahead-of-Time компилятора. Остальные методы интерпретируются.

Компилируемое множество методов задается пользователем.

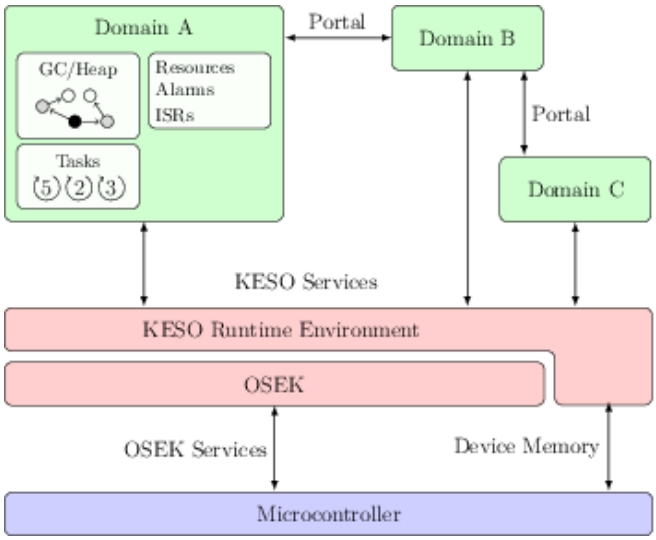
Ниже приведена общая схема архитектуры с сайта проекта[15].



Исходные файлы HVM свободно распространяются и могут быть использованы для решения некоторых задач, но в основном эти задачи имеют учебный характер.

В Университете имени Фридриха-Александра в Эрлангене и Нюрнберге развивают среду разработки на основе Java – KESO [3][4],[19].

Ниже приведен рисунок из [4].



Вот как описывают KESO на сайте ее разработчиков[5].

Система KESO предоставляет возможность запуска одновременно несколько JVM (Multi-JVM) на одном устройстве, для каждой задачи предназначена своя виртуальная машина.

Данная Multi-JVM построена на OSEK/VDX или AUTOSAR OS операционных системах, которые позволяют реализовать механизмы планирования, синхронизации для поддержки многозадачности на одном микроконтроллере.

Система KESO разработана для статических встраиваемых систем. Для таких систем есть возможность применять методы анализа, с помощью которых можно создать систему высоко-адаптированную под приложения Java. Т.к. не требуется динамическая загрузка классов Java в такой системе, то размер результирующего кода может быть очень небольшим. Вместо интерпретации байт-кода на микроконтроллере, Java байт-код компилируется в код целевой платформы.

Приложения KESO разрабатываются на Java и используют унифицированную модель программирования.

KESO предоставляет концепцию, похожую на концепцию процессов в современных операционных системах для персональных компьютеров. Вместо процессов в KESO есть домены, которые

позволяют безопасно сосуществовать множеству задач на одном и том же микроконтроллере. Домены взаимодействуют с помощью унифицированного механизма, работающего похожим образом как Java Remote Method Invocation (RMI) или Remote Procedure Calls (RPC).

KESO позволяет делать программную репликацию критически важных приложений. Копии изолированы друг от друга с помощью доменов. При сбое работы приложения, вместо него может быть использована реплика.

KESO имеет экспериментальную поддержку сети, которые позволяют размещать домены на разных сетевых узлах. Эти домены имеют возможность взаимодействовать, как и домены на одном микроконтроллере.

Уникальные особенности KESO, позиционируемые ее разработчиками:

- первая Multi-JVM для встраиваемых систем;
- предоставляются OSEK/VDX API вызовы и системные примитивы (Java) для разработчика, включая защиту сервисов не предоставляемую OSEK/VDX операционными системами;
- есть возможность разрабатывать драйверы аппаратуры на чистом языке Java;
- предоставляется настраиваемое для каждого домена управление кучей (сборка мусора).

Сравнение двух технологий разработки на базе Java, HVM и KESO можно найти в статье [20]

Две системы разработки программного обеспечения для встраиваемых систем, HVM и KESO, постоянно развиваются и совершенствуются, но все же далеки пока от применимости на

практике и по большей части являются полигоном для исследований в образовательных целях. Общая проблема у этих систем — отсутствие единой удобной среды разработки и некоторые недостатки реализации Java вроде использования интерпретаторов и необходимости использования операционных систем.

Существуют еще другие проекты JVM для встраиваемых систем, например JamVM[24] JamaicaVM[25], Cacao JIT[23], а также Ahead-of-Time компиляторы Java → C вроде FijiVM[26]. Недостаток всех этих систем в том, что в основном это исследовательские проекты, которые не могут считаться в полной мере средами разработки.

2.3 Проблематика современных подходов разработки программного обеспечения на базе Java и .NET для встраиваемых систем.

В предыдущих разделах были рассмотрены средства разработки для встраиваемых систем, основанные на технологиях Java и .NET.

Существует общая проблема в использовании средств разработки на базе Java и .NET применительно к встраиваемым системам.

Дело в том, что самые распространенные процессоры во встраиваемых системах – это системы на кристалле со встроенной памятью RAM. Объем этой памяти за последние десять лет вырос совсем незначительно, с 32к до 128к в среднем. При этом на ближайшую перспективу не видно предпосылок для существенного увеличения объема.

Вместе с тем программные продукты имеют тенденцию к усложнению и соответственно к использованию большего количества вычислительных ресурсов.

Самый важный вычислительный ресурс для таких задач –

оперативная память.

Платформы разработки для встраиваемых систем на основе Java и .NET должны производить очень эффективную сборку мусора, для того чтобы оперативная память использовалась максимально эффективно.

Очень важным свойством встраиваемой системы является время отклика на реальные события. Классический сборщик мусора может делать непредсказуемым отклик системы на реальные события, а это очень важно для встраиваемых систем. Поэтому эффективная сборка мусора важна еще для предсказуемости поведения встраиваемой системы.

Эти требования ограничивает сферу применимости, рассмотренных сред разработки для встраиваемых систем, поскольку в данных системах используется либо интерпретатор, либо динамический сборщик мусора.

2.4 Среда разработки для встраиваемых систем на базе JME

В мире мобильных устройств, с большими по сравнению с системами на кристалле ресурсами, широко распространена платформа JME. Не последнюю роль в широком распространении сыграла удобная среда разработки, включающая библиотеки на все случаи жизни, а так же популяризация языка Java, изучению которого уделяется внимание практически во всех университетах мира.

Изначально Java платформа разрабатывалась для использования во встраиваемых системах, даже совсем простых бытовых устройствах как утюги и холодильники, но потом использование Java платформы стало повсеместным в десктоп системах и мобильных устройствах, а встраиваемые системы оказались забыты. Основную роль здесь сыграла сложность реализации JVM для встраиваемых систем. Чистая

интерпретация слишком расточительно расходует вычислительные ресурсы у систем на кристалле и пагубно сказывается на энергосбережении.

Подход к использованию Java для встраиваемых систем, который основан на применении статической компиляции к платформе J2ME, не оправдывает себя для маленьких систем, поскольку требует наличия операционной системы, а это слишком расточительно для систем на кристалле.

С учетом мирового опыта, была создана архитектура среды разработки, которая основана на платформе JME для встраиваемых систем с ограниченными ресурсами. Созданную систему разработки, по сути можно было бы именовать Java embedded, такое название характеризует применения данной системы.

При использовании подхода на основе JME, значительно облегчается создание программного обеспечения, в силу того, что в мире существует уже развитая культура программирования с использованием Java. Также существует большое количество разработчиков использующих этот язык и для таких разработчиков не составит труда разрабатывать программного обеспечения для встраиваемых систем без изучения новых сред разработки, особенностей библиотек, GUI и т. д.

В области разработки GUI, становится возможным применение всего множества библиотек, созданных за последние 10 лет.

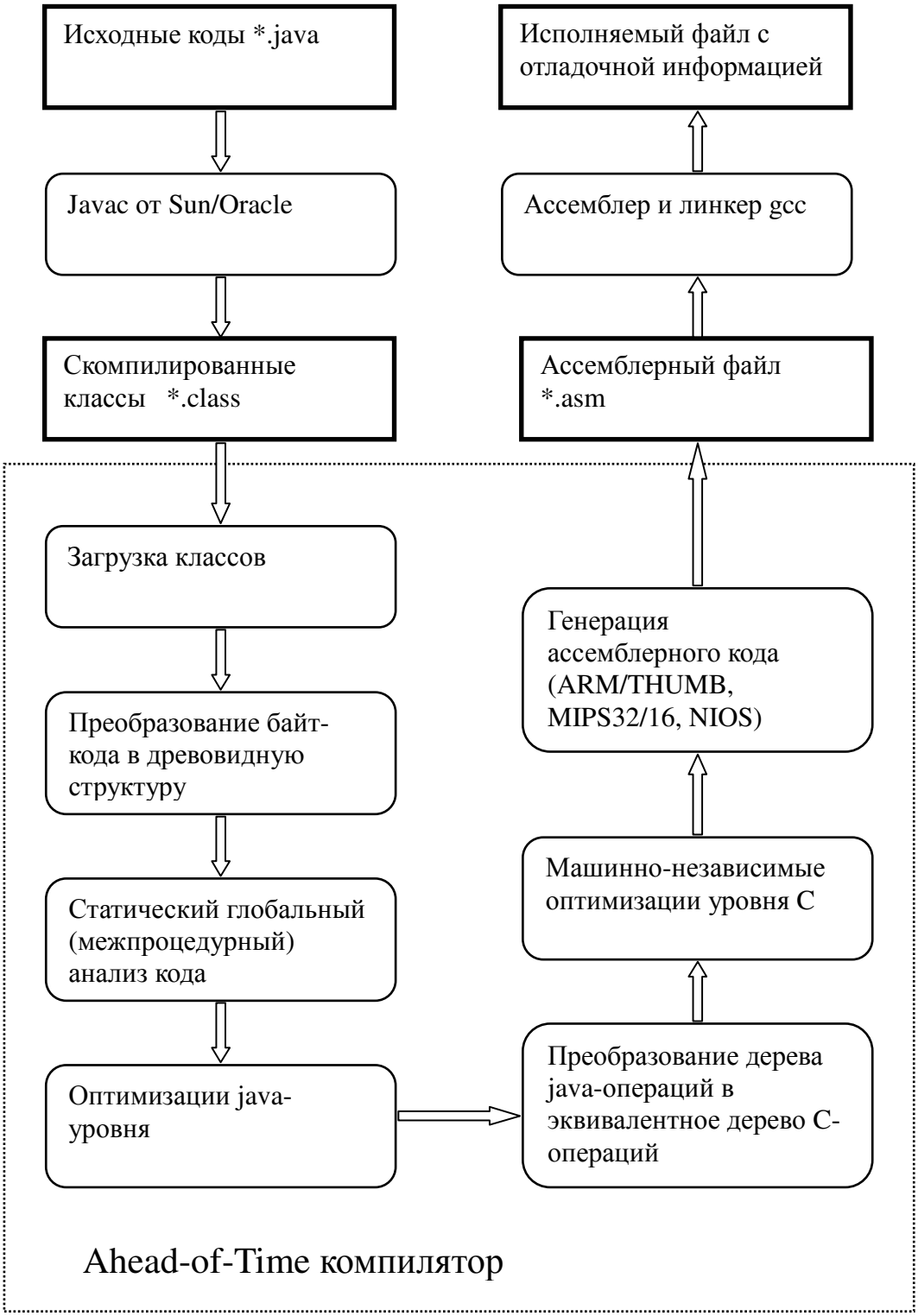
Следует отметить, что при проектировании среды на основе JME был учтено набирающие силу направление в программировании встраиваемых систем. А именно использование свободных IDE, таких как Eclipse и NetBeans. В данном случае была использована NetBeans, как предлагаемая основная среда разработки.

Рассмотрим предлагаемую технологию разработки программного обеспечения для встраиваемых систем с ограниченными вычислительными ресурсами, на основе технологии JME.

В составе среды разработки входит Java Ahead-of-Time компилятор, отладчик, симулятор и библиотеки.

По разработке Java Ahead-of-Time компилятора уже проводились исследования, в частности об этом можно почитать в работе [21].

На рисунке приводится схема работы Ahead-of-Time компилятора, используемого в созданной среде разработки.



Минимальные требования у предложенной технология разработки программного обеспечения:

- объем памяти RAM 10kb;
- разрядность процессора 8/16/32 бит.

Минимальным требованиям системы разработки удовлетворяют большинство систем на кристалле.

2.5 Актуальность моделирования использования памяти.

Как упоминалось ранее, контроль использования памяти во время работы программы, разработанной с помощью одного из современных подходов на базе .NET или Java является очень важным, поскольку от эффективности управления памятью зависит класс задач, которые могут быть решены с помощью встраиваемой системы. В этом отношении проблема использования памяти, также является для актуальной для рассмотренной системы разработки на базе JME.

Одним из способов эффективного управления памятью является статическая сборка мусора. Статическая сборка мусора заключается в определении мест в коде программы, в которых становятся недоступными какие-то объекты, хранящиеся в куче. Такие объекты можно освобождать автоматически, вставляя специальные операции в код программы. Статическая сборка мусора существенно облегчает жизнь динамическому сборщику мусора и позволяет минимизировать задействованные ресурсы памяти. Благодаря эффективной сборке мусора становится возможным сделать предсказуемым отклик встраиваемой системы на реальные события, что очень важно.

Для выполнения статической сборки мусора должно быть проведено моделирование использования памяти.

3. Цель работы

Целью работы, является моделирование использования памяти во время работы программы, разработанной с помощью среды на базе JME.

Результатом исследования по моделированию использования памяти (кучи) должна быть разработанная технология статического анализа, которую может использовать компилятор для выполнения статического управления памятью для среды времени выполнения Java Embedded.

Анализ показал, что задача разработки технологии сводится к следующим подзадачам:

- анализ и исследование существующих алгоритмов статического анализа;
- выбор сочетания алгоритмов;
- выбор метода реализации алгоритмов.

Критериями правильности решения для сформулированной задачи, являются эффективная статическая сборка мусора, высокая скорость статического анализа.

4. Научная новизна

В процессе научно-исследовательской работы над диссертацией, было проведено исследование алгоритмов статической сборки мусора, сформулированы критерии для применимости тех или иных алгоритмов для решения поставленной задачи, выполнен анализ и сравнение алгоритмов на соответствие критериям и выбран оптимальный с точки зрения соответствия критериям алгоритм. Данный алгоритм реализован на практике.

Новаторским подходом в данной работе является использование языка Datalog в алгоритме shape анализа кучи. В других известных работах по тематике статического анализа, которые посвящены использованию Datalog для статического анализа, освещается только применения Datalog в алгоритмах анализа псевдонимов указателей, а например, для shape анализа, такого подхода обнаружено не было.

Проведенное исследование позволило получить метод статического анализа и разработать модель управления памятью (управление сборкой мусора), на основе данного метода среды выполнения для встраиваемых приложений Java.

5. Практическая ценность

Практическая ценность результатов работы заключается в следующем.

Разработана подсистема управления памятью для Java embedded, что в свою очередь делает доступным для большого числа инженеров-разработчиков встраиваемых систем мощного, удобного инструмента разработки программного обеспечения, основанного на технологии Java.

6. Статический анализ программ.

Для программного обеспечения, разрабатываемого для встраиваемых систем, очень важно уметь делать статический анализ программ, особенно на предмет использования памяти необходимой для работы программы.

Вот некоторые примеры использования такого рода анализа:

- в Java приложении можно условно выделить два типа объектов, короткоживущие типа строковых буферов при сложении строк и долгоживущие, короткоживущие объекты можно сразу освобождать;

- можно убирать synchronized методы, если статический анализ показал, что объект не выходит за рамки нитей;

- объекты можно создавать в стеке, если объект не выходит за рамки метода;

- виртуальные вызовы можно заменять статическими;

- возможно применение статического анализа в области security, но стоит отметить, что сфера таких применений не очень широка в области embedded.

Рассмотрим подробнее направления статического анализа программ, наиболее актуальных для технологии Java. Основные направления хорошо сформулированы в работе [2] и приведены ниже.

Анализ псевдонимов указателей

Анализ указателей в программе является одной из важнейших областей статического анализа программ. Он состоит из вычисления статической аппроксимации всех данных, на которые может указывать переменная-указатель или выражение во время выполнения программы. Этот вид анализа составляет базу для практически всех других анализов программы и тесно взаимосвязан с таким механизмом как конструирование call-графа, т. к. значения указателей определяют те цели динамически определяемых вызовов методов в объектно-ориентированных языках или функциональные лямбда приложения.

Указатели присутствуют практически во всех значимых языках программирования. Особенно сложным является анализ указателей для программ на языке С. Указателю в программе на языке С, можно например, присвоить целочисленное значение и это обстоятельство затрудняет анализ.

Указатели в языке Java фактически являются ссылками, над ними не могут быть произведены арифметические операции, и они могут указывать только на начало объекта.

Анализ псевдонимов указателей должен быть межпроцедурным, потому что необходимо убедится в том, что содержимое переменных указателей не меняется в какой-либо процедуре.

В языке С, на анализ накладывается дополнительная сложность из-за возможности косвенного вызова функции по указателю и нужно уметь анализировать функции, которым передается управление в результате таких вызовов.

В языке Java из-за его объектной ориентированности, вызовы виртуальных функций оказываются косвенными. В результате вызов какого-либо метода у переменной - экземпляра объекта, могут быть вызваны разные методы, в зависимости от того, к какому типу принадлежит экземпляр объекта. Более точные знания о типе, а в

идеальном случае точное значение типа, позволяют установить во время компиляции метод, которому будет передаваться управление и передавать его напрямую, это позволяет провести дополнительные оптимизации.
В качестве примера, метода реализации данного вида анализа, можно привести пример работу [28]. В работе для проведения анализа используется реализация Datalog на основе BDD (Binary Decision Diagram).

Анализ структур данных — shape анализ.

Анализ структур данных (shape анализ) является разновидностью статического программного анализа и предназначен для нахождения и анализа связных динамических структур данных, используемых программой в процессе выполнения.

Shape анализ является по сути формой pointer анализа, но с его помощью можно получить более точные результаты.

Целью shape анализа, является получение структуры кучи, которая будет использоваться программой во время выполнения.

Связность

В этом типе анализа рассматриваются два свойства данных — достижимость и пересечение. Эти свойства имеют важное применение в преобразованиях для достижения параллелизма. Например, рекурсивный вызов или цикл может быть преобразован во множество нитей работающих параллельно. Свойство достижимости представляет собой предикат, который говорит, существует ли из области памяти m1 в область памяти m2 путь из указателей? А свойство пересечения может быть выражено предикатом, который говорит о том, существует ли путь из области памяти m1 в область m3 != m1 и путь из m2, m2 != m3?

Анализ регионов памяти.

Существует методы анализа, цель которых заключается в идентификации регионов памяти, содержащих одну и ту же структуру данных. Анализ таких регионов может помочь при сборке мусора, например, становится возможным использование специального пула объектов для создания одной и той же структуры данных в одной и той же секции памяти.

Зависимость по памяти

Часто интерес к shape информации или связности кучи состоит в использовании этих результатов для определения зависимостей между данными по памяти. Например,

в примере

```
x = 1;
x = x+1;
```

мы не можем две строки исполнять параллельно, поскольку есть зависимость по памяти.

Если анализ может определить, что два куска кучи не пересекаются, то мы можем быть уверены, что операции, которые выполняются над одним из кусков кучи, не оказывают влияние на другой кусок кучи. В общем случае было бы желательно, чтобы непосредственно определять, какие части кучи могут быть модифицированы любым оператором программы.

Время жизни объектов

Еще одно важное направление статического анализа — это определение времени жизни объектов. Информация о времени жизни объекта позволяет оптимизировать управление памятью программы. Становится возможным собирать неиспользуемые объекты, при этом, не прибегать к сборщику мусора, таким образом можно экономить вычислительные

ресурсы программы.

Из приведенных выше направлений статического анализа программ, наиболее интересные направления анализа применительно к embedded Java – это анализ псевдонимов указателей и shape анализ.

7. Обзор методов статического анализа программ.

7.1 Использование Datalog для анализа псевдонимов указателей

Теоретические основы анализа псевдонимов указателей хорошо раскрыты в классической книге по компиляторам [31], а практическая реализация выполнена в работах [27] ,[28], [29]. Ниже приводятся теоретические выдержки из книги [13].

Граф вызовов.

Граф вызовов представляет собой множество узлов и ребер, такое что

- каждой процедуре программы соответствует один узел;
- каждой точке вызова (место в программе, где происходит вызов) соответствует один узел в графе;
- если точка вызова c, может вызывать процедуру p, то в графе имеется ребро от узла c к узлу p.

Если процедура вызывается непосредственно, то целевой код каждого вызова может быть определен статически. В этом случае, каждая точка вызова в графе имеет единственное ребро ровно к одной процедуре. Если же программа включает использование, например указателей на функции, то в общем случае целевой код неизвестен до момента выполнения и точка вызова может быть связана со многим процедурами графа вызовов.
Косвенные вызовы распространены в объектно-ориентированных языках программирования. В частности, при перекрытии методов в подклассе, использование метода m может означать любой из большого количества разных методов, зависящих от конкретного подкласса объекта. Использование вызовов таких виртуальных методов означает, что мы должны знать тип объекта до того, как сможем определить, какой именно метод будет вызван

В общем случае, наличие ссылок или указателей на функции или методы требуют от нас статической аппроксимации потенциальных

значений всех параметров процедур, указателей на функции и типов объектов, получающих сообщения. Для выполнения точной аппроксимации требуется применение межпроцедурного анализа. Этот анализ итеративен и начинается со статически определяемого целевого кода. При обнаружении нового целевого кода, анализ добавляет в граф вызовов новые ребра и повторяет выявление нового целевого кода, пока данный процесс не сойдется.

Контекстно-нечувствительный анализ.

Межпроцедурный анализ очень сложен, в том числе потому, что поведение каждой процедуры зависит от контекста, в котором она вызвана.

Простой, но неточный подход к межпроцедурному анализу, известный как контекстно-нечувствительный анализ(context-insensitive analysis), заключается в рассмотрении каждой инструкции вызова и возврата как операции безусловного перехода. Мы создаем надграф потока управления, в котором помимо обычных ребер внутрипроцедурных потоков управления, имеются дополнительные ребра соединяющие:

- каждую точку вызова с началом вызываемой в ней процедуры;
- инструкцию возврата с точками вызова.

Добавляются инструкции присваивания каждого фактического параметра соответствующему формальному параметру, а также присваивания возвращаемого значения переменной, получающей результат. Затем можно применить стандартный анализ, предназначающийся для использования в процедуре, к надграфу потока управления для поиска контекстно-нечувствительных межпроцедурных результатов. При своей простоте такая модель абстрагируется от важных взаимоотношений между входными и выходными значениями в вызовах процедур, что приводит к неточности такого анализа

Контекстно-чувствительный анализ указателей.

Контекстная чувствительность может существенно повысить качество межпроцедурного анализа.

Рассмотрим метод контекстно-чувствительного анализа, основанный на клонировании. Такой анализ просто клонирует методы, по одному для каждого интересующего нас контекста. Затем мы применяем к клонированному графу вызовов контекстно-нечувствительный анализ. Хотя этот подход и кажется простым, но проблема заключается в деталях обработки больших количество клонов. Сколько всего существует контекстов? Число всех контекстов очень большое для приложений Java(порядка 10**14).

Большое число различных способов анализа программ, могут быть выражены естественным и простым образом с помощью логических языков программирования, таких как Prolog и Datalog.

Использование логического языка программирования для выражения программного анализа имеет ряд преимуществ. Первое, анализ программ сильно упрощается. Анализ, записанный в нескольких строчках кода на языке Datalog, превращается в тысячи строк кода на традиционном языке. Второе, поскольку вся аналитическая информация выражена единообразно, легко становится использовать результаты анализа или их комбинировать. И к тому же оптимизации в Datalog могут быть применены ко всем типам анализа, выраженным на этом языке.

Datalog

Язык Datalog - это логический язык программирования. Язык Datalog синтаксически является подмножеством Prolog.

Элементами Datalog являются атомы вида p(X1,X2,..,Xn), здесь:

1) p - предикат, символ, который представляет инструкции вроде

”определение достигает начала блока”;

2) X1, X2, .., Xn — аргументы, такие как переменные или константы, в качестве аргументов предиката могут выступать простые выражения.

Основной атом представляет собой предикат, аргументами которого являются константы. Каждый основной факт является утверждением о некотором конкретном факте, и его значением является либо истина, либо ложь.

Часто оказывается удобным представлять предикат отношением или таблицей его истинных основных атомов. Каждый основной атом представлен одной строкой или кортежем (tuple) отношения. Столбцы отношения называются атрибутами, и каждый кортеж имеет компонент для каждого атрибута. Атрибуты соответствуют компонентам основных атомов, представленных отношением. Любой основной атом в отношении истинен, а основные атомы, не входящие в отношение, ложны.

Правила представляют собой способ выражения логических выводов. В Datalog правила служат также для предложения того, каким способом должны быть вычислены истинные факты. Правила имеют вид:

H:-B1&B2&..&Bn.

Основные компоненты правила:

– H является атомом, а B1,B2,...Bn, являются литералами-либо атомами, либо их отрицаниями;
– H - заголовок правила, а B1,B2, .. , Bn образуют его тело;
– каждый из литералов Bi называется также подцелью правила;
– символ :- читается как «если».

Программа на Datalog представляет собой набор правил. Такая программа применяется к «данным», т. е. к множеству основных атомов для некоторых предикатов. Результатом выполнения программы, является множество основных атомов, полученное путем применения правил до

тех пор, пока не будут сделаны все возможные выводы.

Для определения путей в графе можно, например, ввести такие правила:

1) path(X,Y) :- edge(X,Y);
2) path(X,Y) :- path(X,Z) & path(Z,Y).

Бинарные диаграммы решений.

Бинарная диаграмма решений (BDD) - это форма представления булевой функции в виде направленного ацикличного графа. По сути, бинарные диаграммы представляют собой сжатое представлений множества отношений.

Бинарные диаграммы решений широко используются в области программного обеспечения CAD, для синтеза схем, формальной верификации алгоритмов и для анализа программ.

Для представления 2**n булевых функций от n переменных, нет краткого способа представления, однако булевы функции на практике достаточно регулярны, и для них можно найти краткие формы бинарных диаграмм. Булевы функции, использующиеся при анализе программ, так же поддаются описанию в форме BDD.

Диаграммы бинарного выбора представляют булевы функции в виде ориентированного графа с корнем. Каждый внутренний узел взаимно-однозначно соответствует какой-то переменной из булевой функции. В нижней части графа есть два листа со значениями результата функции 0 и 1. Каждый внутренний узел, соответствующий переменной имеет два ребра к дочерним с метками 0 и 1, которые соответствуют значениям, которые принимает переменная.

Одна из реализаций языка Datalog, широко используемая в исследовательских целях - это bddbddb [29]. Эта реализация, использует BDD для представления отношений.

7.2 TVLA, технология shape анализа основанная на трехзначной логике.

Рассмотрим в данном разделе еще одну методику статического анализа, актуальную с точки зрения ее применимости к платформе JME.

Система TVLA предназначена для автоматической генерации алгоритма анализа из операционной семантики какой-либо программы. Система основана на анализе с использованием трехзначной логики TVLA(Three-Valued-Logic Analysis engine).

Операционная семантика программы записывается в специальной форме, основанной на предикативной логике первого порядка с транзитивным замыканием. Дополнительными входными данными для TVLA является абстрактное представление всех возможных состояний памяти в начале анализируемой программы. Система TVLA автоматически генерирует абстрактную семантику и, для каждой точки программы продуцирует консервативное абстрактное представление состояний памяти в этой точке.

$\wedge$	0	1	1/2
0	0	0	0
1	0	1	1/2
1/2	0	1/2	1/2

$\vee$	0	1	1/2
0	0	1	1/2
1	1	1	1
1/2	1/2	1	1/2

$\neg$	
0	1
1	0
1/2	1/2

В таблице представлена 3-хзначная интерпретация Клини логических операторов.

С помощью 3-хзначной логики удается получить более компактное и

выразительно представление для состояний памяти программы и в сфере статического анализа система TVLA больше всего подходит для shape анализа.

TVLA можно задействовать для анализа таких групп приложений:

- односвязный список;
- двусвязный список
- программы сортировки и др.

Система TVLA представляет собой набор программ для выполнения анализа. Эту систему можно на практике задействовать при статическом анализе программ. Вместе с тем система обладает существенным минусом — высокими требованиями к вычислительным ресурсам и длительным временем обработки данных.

8. Основная технология статического анализа программ для платформы JME

Технологии статического анализа программ Java, рассматриваемые в предыдущей главе, а также многие другие существующие и описанные в литературе, в основном применимы к каким-то отдельным областям анализа и целостного подхода ко всем аспектам анализа у них нет.

В качестве основного подхода для моделирования работы кучи, был выбран комплексный подход, аналогичный тому, который описан в докторской диссертации Марка Маррона [30].

С теоретической точки зрения, технология моделирования кучи из работы Марка Маррона, в применении к нуждам статического анализа Java embedded, выглядит предпочтительнее других работ в этой области в силу своей комплексности.

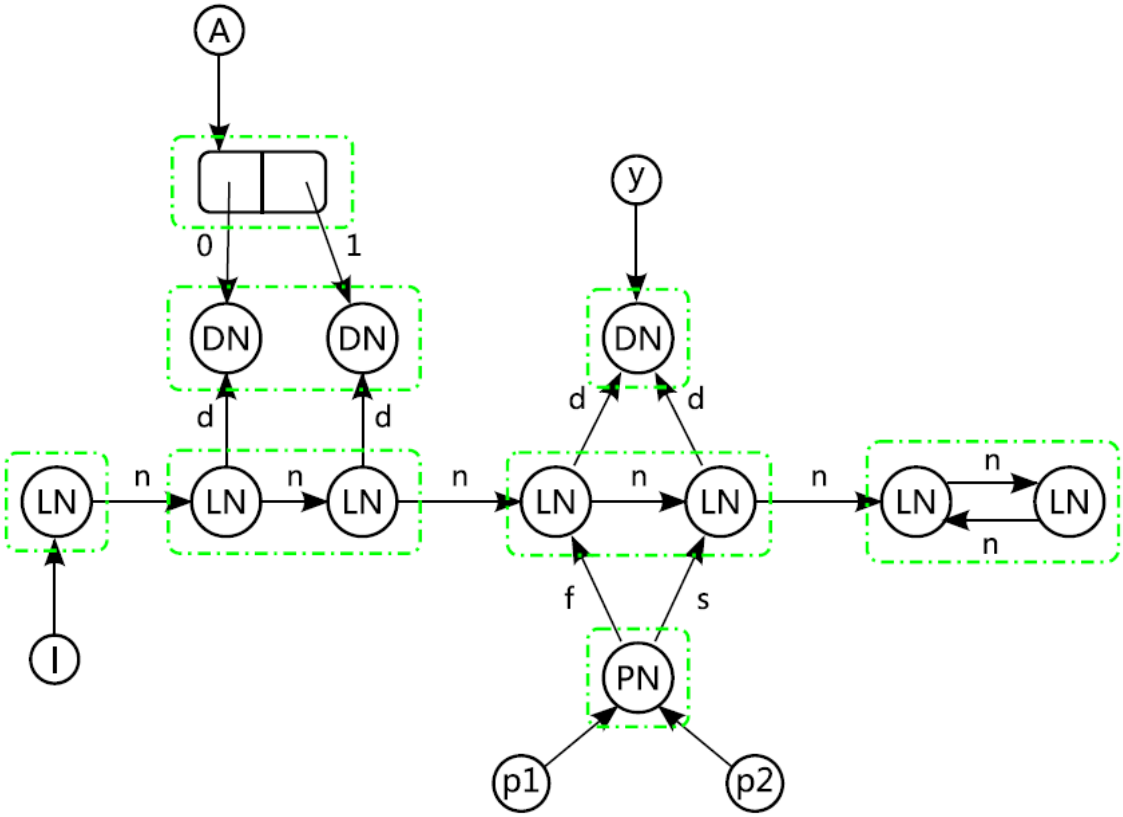
Рассмотрим основу метода статического анализа Java приложений, применяемого в работе. Основой метода анализа является граф, описывающий модель кучи программы. Ниже приводится описание этой структуры из работы [30].

Семантика языка программирования определяется состоянием, которое отображает переменные на значения и хранилищем, которое отображает адреса на значения. Состояние и хранилище вместе рассматриваются как состояние конкретной кучи. Для моделирования такой кучи, используется направленный мульти-граф с метками ((V)variables, (O)bjects, (R)eferences), где каждая переменная v V это переменная из состояния, каждый o O является объектом в хранилище, а каждая направленная помеченная дуга r R представляет собой ссылку (указатель между

объектами или ссылку на переменную).

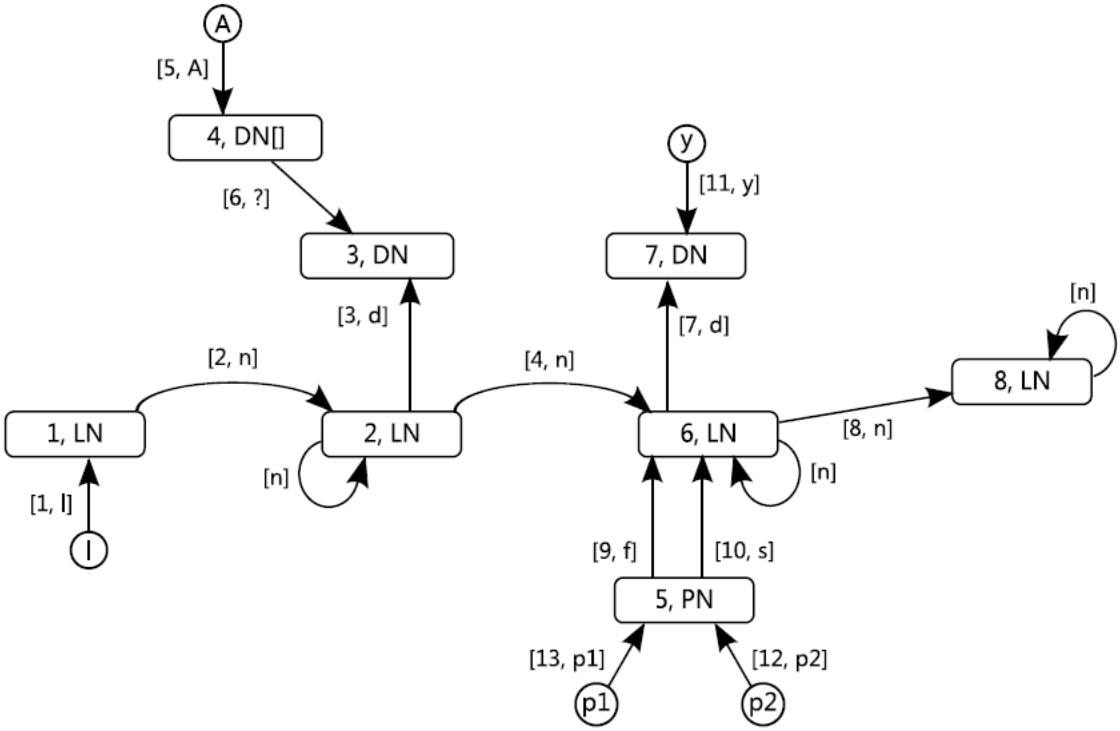
Регионом памяти называется подмножество объектов в памяти, вместе со всеми указателями, соединяющими эти объекты и со всеми ссылками между регионами, которые начинаются или заканчиваются на объекте в регионе.

Ниже показан пример кучи из работы [30].



В примере используется несколько структур данных DN={int val}, класс для построения списков LN={DN data, LN next} и класс PN={LN first, LN second}. Переменная l указывает на голову связного списка из LN элементов. Данные объединены пунктирными линиями в регионы памяти.

С помощью абстракции объектов в регионах вместе с узлами и указателями между регионами мы получаем Labeled Storage Shape Graph (LSSG) модель на следующей иллюстрации



Каждый узел в LSSG и дуга на другой узел, получает уникальный номер и помечается типом объектов из региона (или именами переменных). Дуги также получают уникальный числовой идентификатор и помечены смещением в хранилище, где хранится ссылка (идентификатор поля, имя переменной или специальное смещение для указателей, хранящихся в массиве). Единственная информация, сохраненная в этой модели - это типы объектов в регионах и некоторая грубая информация о соединениях, определяемая структурой графа и смещениями в хранилище, привязанными к дугам.

В такой модели пока еще не достаточно средств, для более точного выражения свойств кучи. Для эффективного выполнения оптимизаций нужно расширить свойства модели.

Расширение LSSG.

Граф LSSG можно дополнить новыми свойствами и получить расширенный LSSG (Extended Label Storage Shape Graph). Перечислим

свойства расширенного LSSG.

1) Базовые инструментальные свойства.

1.1) Типы.

Каждый узел в графе представляет собой регион кучи (который может состоять из объектов разных типов). Это множество типов отображается в узлах.

1.2) Смещения.

Каждая дуга в графе представляет собой переменную или множество указателей хранимых в поле объекта или массиве. Дуги помечены смещением в хранилище, где хранится объект по ссылке.

Для массивов и коллекций используются специальные имена для указания местоположения.

1.3) Линейность.

Свойство линейности отражает количественную информацию и имеет два возможных значения:

- значение 1, которое говорит, что узел (дуга) содержит 0 или 1 объектов (указателей);
- значение ω, которое говорит, что узел (дуга) содержит любое количество объектов (узлов) от 0 до бесконечности.

2) Абстрактное представление.

В узлах содержится свойство, определяющее вид региона {(S)ingleton, (L)ist, (T)ree,(M)ultipath, (C)ycle}

- если регион имеет представление Singleton, то это означает, что он не связан с другими регионами;
- если регион имеет представление List, то это означает, что регион имеет структуру списка;
- если регион имеет представление Tree, то это означает, что регион имеет структуру дерева;

- если регион имеет представление Cycle, то это означает, что регион имеет структуру цикла;
- если регион имеет представление Multi-Path, то это означает, что в регионе между какими-то двумя объектами есть два разных пути.

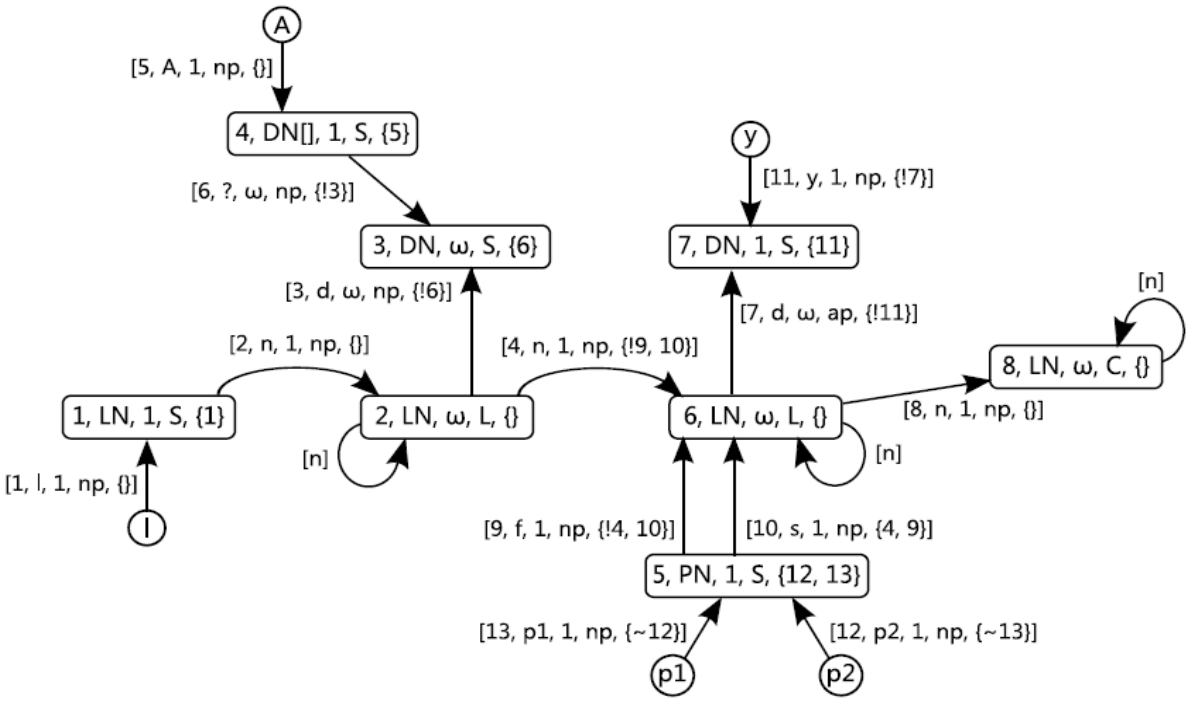
3) Связность и интерференция.

Связность предназначена для обозначения отношений достижимости между ссылками, которые определяются различными дугами в модели кучи, для выражения связности используются такие значения - {share,connected, disjoint}.

Интерференция используется для отражения отношений достижимости между ссылками, определяемыми одной и той же дугой в модели кучи, для выражения интерференции используются такие значения - {aliasing pointers(ap), interfering pointers(ip), non-interfering pointers(np)}.

4) Доминирование.

С помощью новых свойств мы для моделирования и анализа кучи получаем новый, расширенный LSSG граф. На иллюстрации приводится та же структура кучи, что и на предыдущих двух рисунках, но в новой интерпретации с помощью расширенной версии LSSG:



Как уже упоминалось, в основе статического метода моделирования и анализа кучи для платформы JME embedded, было решено использовать методику из работы [30], основанную на использовании расширенной версии Labeled Storage Shape Graph (LSSG), которая описана выше.

Не смотря, на то, что теоретическая часть статического анализа из работы[30] проработана хорошо, есть некоторые трудности с практической реализацией формальных методов анализа на практике. В основном эти трудности связаны с вычислительной сложностью задач, а также с ограниченностью вычислительных ресурсов, т. к. анализ требует хранения большого объема информации в оперативной памяти.

В качестве метода реализации структур данных и операций над ними, предложен комплексный подход, использующий язык Datalog для представления отношений между блоками и непосредственное

представление на Java и показаны преимущества такого подхода

Следует отметить, что Datalog, широко используется в различных работах по статическому анализу Java программ. В основном эти работы относятся к анализу псевдонимов указателей, но не встречается работ, где Datalog применялся бы к статическому shape анализу. Поэтому можно говорить о том, что подход использующий Datalog для проведения статического shape анализа Java программ, является новаторским.

В качестве реализации Datalog был выбран пакет bddbddb.

9. Сравнение методов реализации алгоритма статического анализа кучи для системы разработки на базе JME.

В обзорной части методов статического анализа были рассмотрены анализ указателей, метод shape анализа TVLA, а так же метод моделирования кучи Марка Маррона [30].

Статическая сборка заключается в определении точек кода в программе, где становятся недоступными какие-либо объекты, расположенные в куче и автоматическом удалении этих объектов.

Для статической сборки мусора один лишь анализ указателей не подходит, поскольку результат анализа не сообщает информацию о виде структур данных в построенной модели кучи. Чтобы автоматическая сборка мусора была эффективна, нам необходимо уметь определять деревья связанных между собой структур, которые стали недоступными в данной точке кода программы.

Цель shape абстракции заключается в том, чтобы сделать анализ указателей более точным. Можно сказать, что анализ указателей предоставляет информацию, о том, может ли у нас указывать объект на какую-то область памяти. В то время как, во-первых, shape анализ дает ответ на вопрос о том, какой объект указывает на каждый элемент кучи именно в данной точке программы. Во-вторых, благодаря атрибутам в shape графе, мы можем выбирать оптимальный способ освобождения памяти, простую операцию free или сложный деструктор. Благодаря этим свойствам, shape анализ может использоваться для автоматической сборки мусора в программе.

В общем случае статическая сборка мусора осуществляется

схематически таким образом.

Здесь используется классический анализ времени жизни переменной, который есть в любом компиляторе.

В ходе сравнительного анализа методов shape анализа, в качестве основного метода статического анализа был выбран метод [30].

Метод [30] описан в основном теоретически, не смотря на то, что авторы методы произвели его проверку на тестовых примерах, для целей реальной статической сборки мусора, метод применен не был. Поэтому требуется еще решить задачу эффективной программной реализации этого метода.

Опишем общую схему алгоритма shape анализа

1) На основе анализа вызова методов создается call граф программы. При этом рекурсивные вызовы и циклы объединяем в отдельный узлы, таким образом, граф получается ациклическим.
2) С помощью call графа находим все возможные пути в программе и таким образом получаем все множество контекстов исполнения в программе.
3) Производим построение shape графа для каждого контекста в программе и для каждого блока кода. Построение shape графа

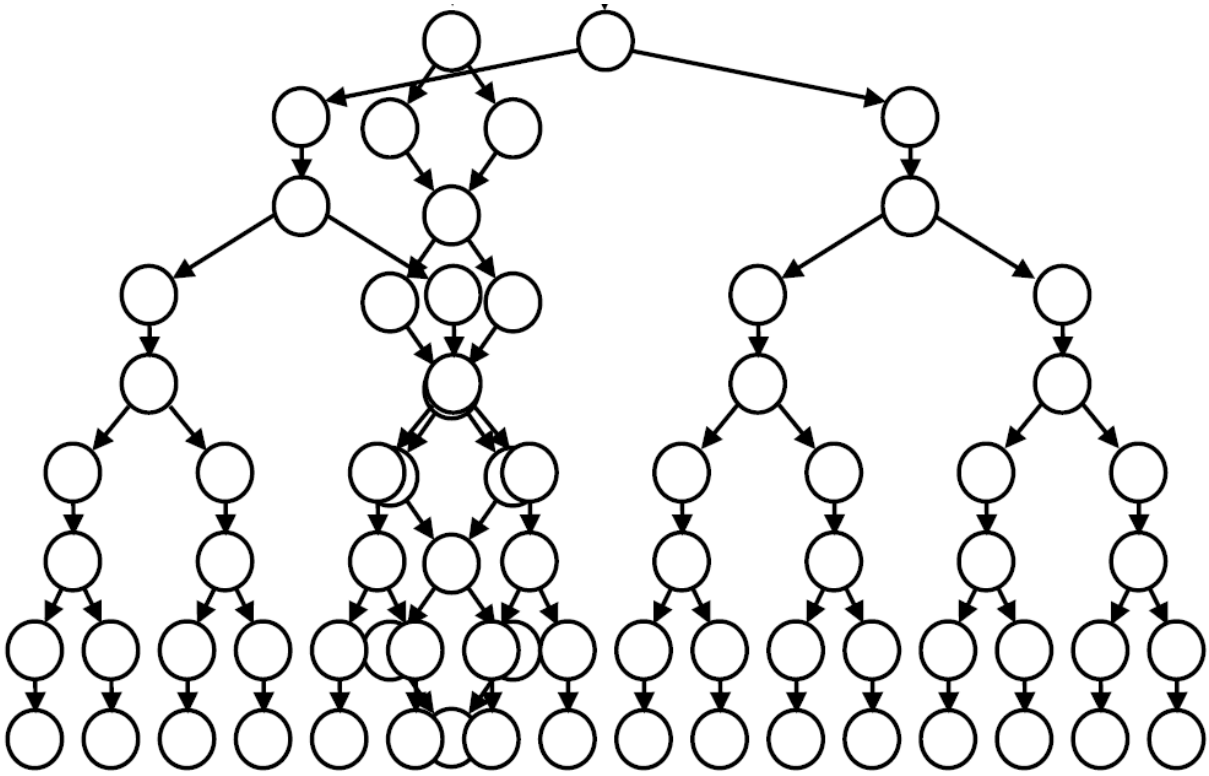
представляет собой итеративный процесс, процесс заканчивается, когда находится fix-point, т.е. граф перестает меняться. Можно выделить два уровня в построении графа.

- Внутриблочный уровень. В каждом блоке, итеративно строим shape граф на выходе из каждого блока.
- Внутрипроцедурный уровень. Пробегая по блокам, выполняем построение shape графа внутри блоков, пока отношения между блоками не перестанут меняться.
- Межпроцедурный уровень.

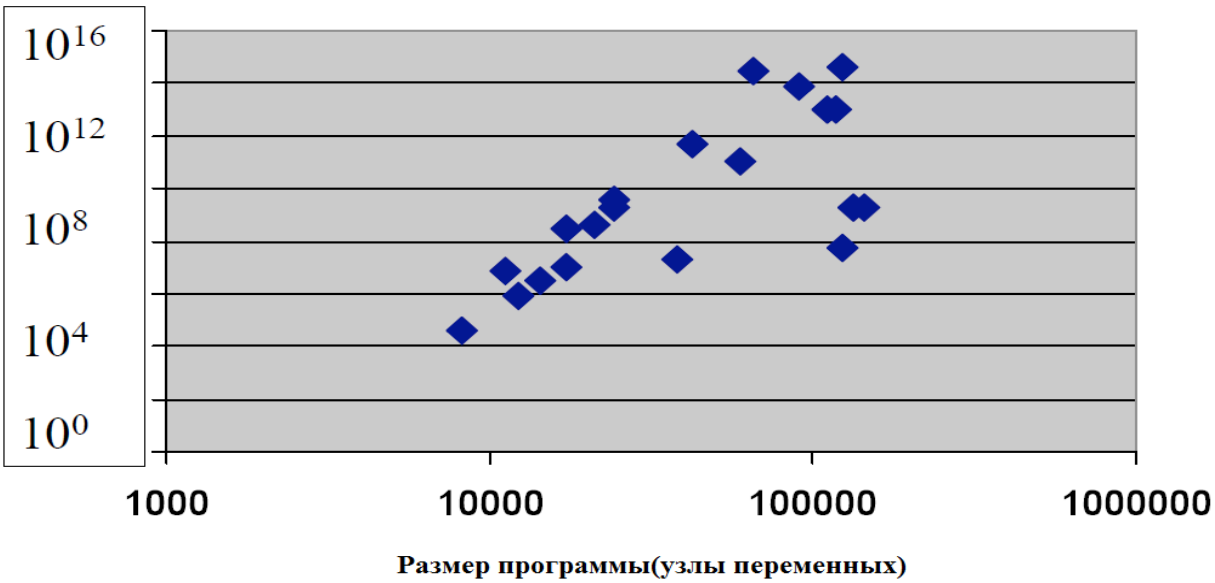
4) Объединяем shape графы для контекстов выполнения программы.
5) Получаем после объединения контекстов для каждого блока свой shape граф.

Существует проблема в shape анализе, связанная с необходимостью учитывать контексты программы, так как shape анализ является расширением pointer анализа с учетом возможных путей выполнения программы.

Даже если в программе нет рекурсии, то количество возможных путей в программе растет экспоненциально, как на иллюстрации из работы [38]



На следующей диаграмме приводится количество контекстов программ для 20 популярных Sourceforge Java приложений из работы[38]



Эффективным способом представления огромного пространства состояний и работы с ним является использование Binary Decision Diagram для представления контекстно-чувствительных отношений.

Различные контексты содержат в себе одинаковые отношения, которые могут объединяться с помощью BDD. Таким образом, происходит сжатие всего пространства контекстов и с таким представлением удобно работать

Кроме BDD существуют другие способы представления информации для анализа, такие как фреймворк Doop[37] и др.

Учитывая возможные варианты представления информации для анализа, а также того, что анализ может меняться и расширяться, появляется необходимость создания промежуточного слоя абстракции, с помощью которого можно было бы удобно описывать различные виды статического анализа и представлять графы с атрибутами. Так же важно, чтобы была возможность отображать абстрактный слой на различные эффективные представления, которые в принципе могут меняться от задачи к задаче.

В качестве слоя абстракции было выбрано сочетание формы представления с использованием Datalog и непосредственной формы представления на Java.

Сравнение форм представлений shape графа

Типичные операции shape анализа	Непосредственная форма	Datalog форма
Нормализация LSSG	Легко реализуется и обрабатывается	Производится много операций добавления новых отношений и удаления старых отношений, неэффективно реализуется

Обработка информации внутри блоков операций	Легко реализуется и обрабатывается	Производится много операций добавления новых отношений и удаления старых отношений, неэффективно реализуется
Обработка информации между блоками операций	Ввиду множества программных контекстов, обрабатывается неэффективно	Легко реализуется в силу монотонности операции, то есть просто добавляются новые отношения к старым
Скорость операций сравнения графов	Низкая, выявляется изоморфизм подграфов.	Высокая, сводится просто к сравнению указателей

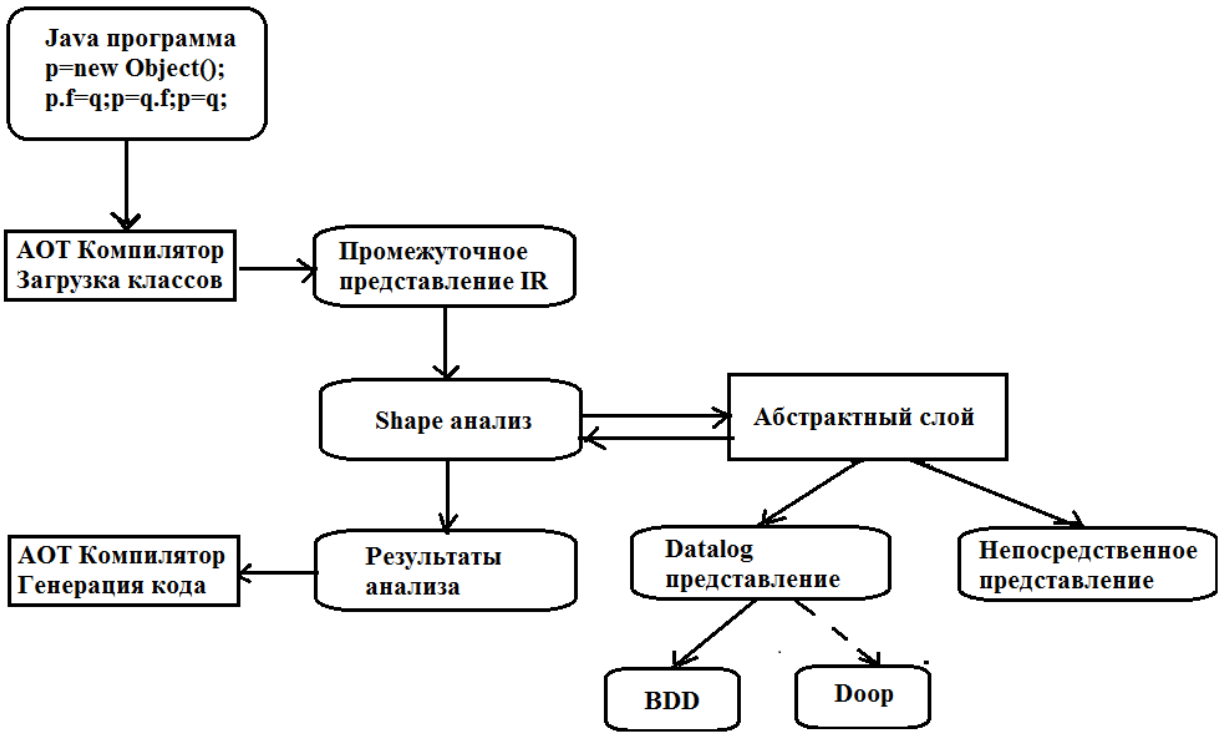
Из таблицы видно, что та или иная форма представления для различных операций shape анализа, имеет свои недостатки и преимущества. Поэтому является целесообразным использовать сочетание этих двух форм представления.

Используя различные представления для различных стадий анализа, можно получить более высокую скорость выполнения анализа в целом

Подобный способ представления анализа используется в других исследовательских работах, например в работе [39]. В этой работе используется понятие фазо - чувствительного представления состояний для реализации TVLA shape анализа. Все структуры там изначально являются изменяемыми и могут обновляться с помощью

применения TVLA. В последующем, они становятся неизменяемыми и могут сравниваться на равенство с другими структурами, но не могут обновляться. Для реализации такого подхода используется эффективное по времени представление для обновления изменяемых структур и эффективное по использованию памяти представление для неизменяемых структур. Методы реализации таких представлений в работе [39] предложены, но больше отвечают особенностям TVLA.

Как отмечалось в предыдущих разделах, Datalog широко используется, например, для анализа указателей, но для shape анализа пока применений в литературе не встречается, поэтому можно считать такой подход новаторским. Схема окончательной реализации показана на рисунке



10. Тестовые результаты и диаграммы.

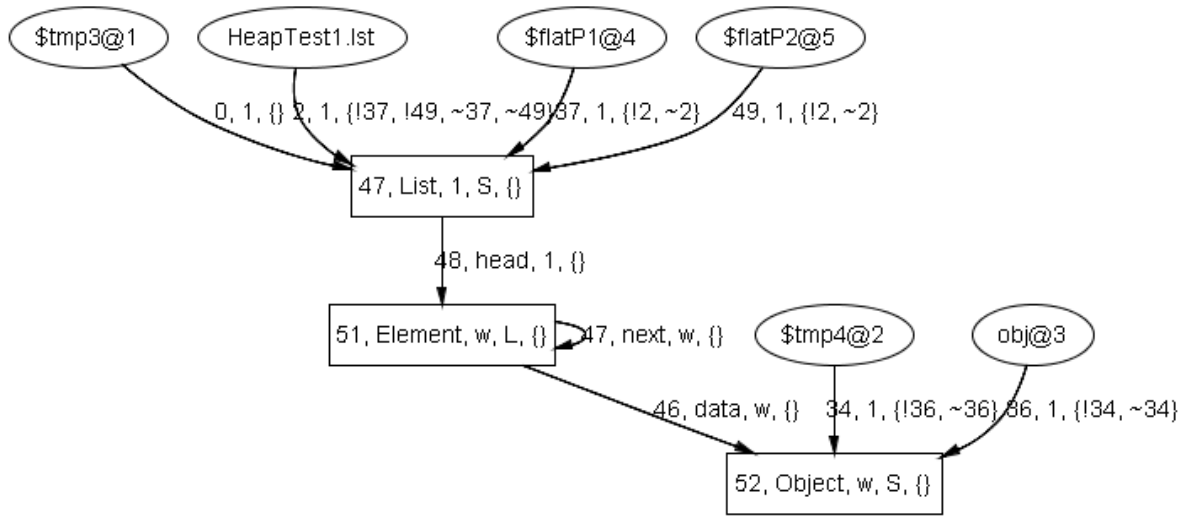
Итак, для реализации shape анализа из работы [30] был выбран комплексный подход, использующий логический язык Datalog для представления отношений между блоками кода и непосредственное представление на Java для анализа внутри блоков кода.

Для демонстрации результатов статического анализа и построения графа LSSG рассмотрим такой пример.

```
class Element
{
  Object data = null;
  Element next;

  public Element(Object d, Element n)
  {
    this.data= d;
    this.next= n;
  }
}

class List
{
  Element head;

  public List()
  {
  }

  public void add(Object e)
  {
    head= new Element(e, head);
  }

  public List reverse()
  {
    List list= new List();
    Element elem= head;
    while (elem != null)
    {
      list.add(elem.data);
      elem= elem.next;
    }
    return list;
  }
}
```

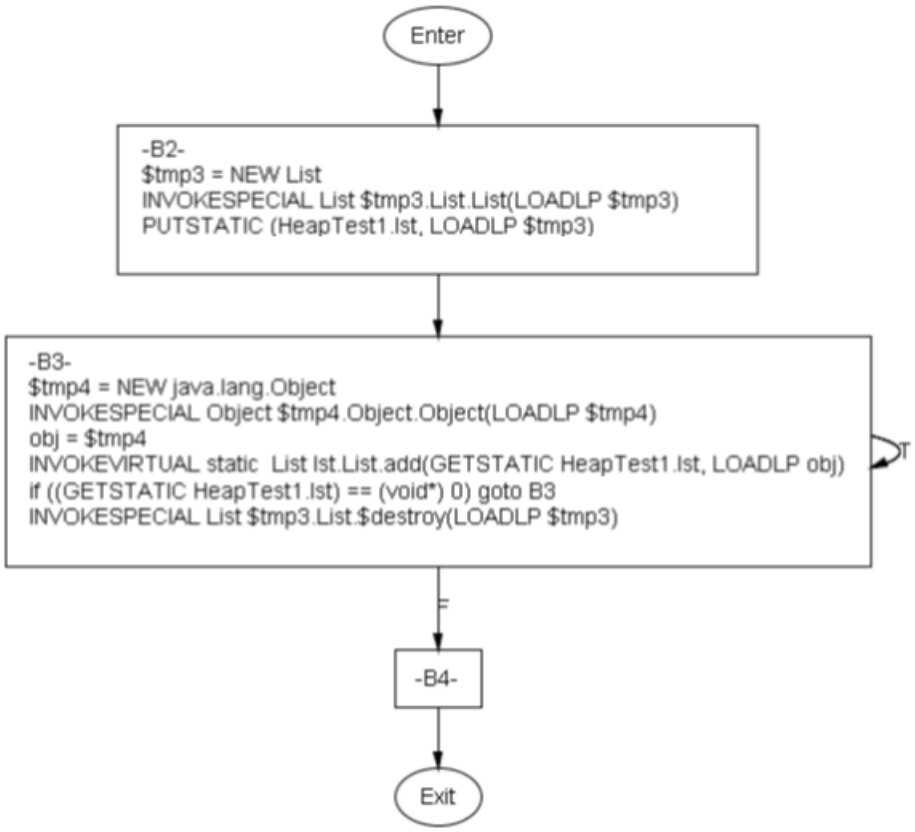
```
…
 public void test()
 {
  lst= new List();
  do
  {
   Object obj = new Object();
   lst.add(obj);
  }
  while (lst == null);
 }
```

Для данного примера, анализ построил такой граф LSSG. Узел 51 в графе соответствует списку из объектов Element, а узел 41 объекту List в единичном экземпляре.



После того, как отработал статический анализ, компилятор обнаружил место, где объекты становятся недоступными и их можно удалить.

Рассмотрим преобразованный байт-код на иллюстрации.



Здесь переменные tmp3, tmp4 являются временными переменными, генерируемыми компилятором Java.

На диаграмме видно, что по результатам анализа, в конце узла B3 была вставлена операция вызова метода, освобождающего память, занимаемую объектом-списком автоматически:

…..

INVOKESPECIAL List$tmp3.List$destroy(LOADLP $tmp3)

…..

Чем полнее представлена информация с помощью атрибутов LSSG shape графа, тем эффективней можно выполнять сборку мусора в программе. Например, с помощью вставки операции free в код программы мы можем освобождать не только отдельный объект, который в результате анализа определился как неиспользуемый, но и

целые подграфы связанных объектов.

Рассмотрим далее результаты проверки работы статического анализа на различных тестовых наборах.

Результаты тестирования статического анализа на типовых операциях, встречающихся при разработке GUI:

Тест	Описание	Классы	Методы	Байткод	Размер задействованной кучи	Размер статически собранного мусора
1	Смена формы GUI на экране	25	60		12к	6к
2	Открытие/закрытие диалога с combobox	30	72		14к	7к
3	Обработка строковых операций	10	28		9к	8к
4	Создание/удаление окна с картинкой	35	45		20к	19к

Результаты тестирования статического анализа на наборе алгоритмических тестов:

Тест	Описание	Классы	Методы	Байткод	Размер задействованной кучи	Размер статически собранного мусора
1	Быстрая сортировка				20к	12к
2	Удаление элемента списка				22к	15к
3	Удаление поддерева в дереве				40к	30к

Результаты тестирования статического анализа на тестовом наборе Java приложений:

Тест	Описание	Классы	Методы	Байткод	Размер статически собранного мусора %
freets	Система синтезирования речи	215	723	48к	40,00%
jetty	Чат клиент	283	993	61к	45,00%
joone	Java фреймворк для нейросетей	375	1531	92к	45,00%
sshdaemon	SSH демон	485	2053	115к	30,00%
sshterm	SSH терминал	808	4059	241к	55,00%

11.Основные результаты и выводы

Выполнен комплекс теоретических, практических и исследовательских работ по выбору метода статического анализа для моделирования поведения кучи во время работы программы. Благодаря проведенным работам был выбран метод для решения проблемы статической сборки мусора в среде выполнения программ Java embedded.

Выбранный метод был опробован на практике. Проанализировано поведение системы управления памятью для различных вычислительных задач.

Результаты тестирования системы управления памятью можно признать положительными, т.к. они удовлетворяют основным критериям по эффективности, скорости работы.

Дальнейшие исследования. Реализованный метод статического shape анализа значительно упростил сбор мусора во время работы программы. Однако потенциал статического shape анализа не исчерпан полностью. Путем введения новых атрибутов графа LSSG и дополнительным анализом можно выявлять еще больше объектов, которые можно удалять статически, тем самым сведя к минимуму работу динамического сборщика мусора. В пределе статический анализ должен собирать весь мусор и можно будет отказаться от динамического сборщика мусора совсем. Данная задача будет являться темой дальнейших исследований.

12. Практическое использование результатов.

Разработанная технология статического анализа была применена в не больших проектах для встраиваемых систем с ограниченными ресурсами. Статическая сборка мусора показала свои преимущества перед одной только динамической сборкой.

Благодаря достигнутым в ходе исследования результатам, среда разработки на базе JME для встраиваемых систем получила большую гибкость. Стало возможным использовать данную среду для решения более широкого круга задач, в частности задача связанных с использованием GUI на основе Java во встраиваемых системах. В перспективе можно ожидать, что благодаря полученным преимуществам, круг пользователей среды разработки на базе JME будет расширяться за счет популярности платформы Java и связанного с этим легкого перехода на новую технологию, облегчения и удешевления разработки.

Список используемой литературы

1. Jens Kühner. EXPERT .NET MICRO FRAMEWORK , 2009.
2. Sun Microsystems. J2ME Building Blocks for Mobile Devices, 2000.
3. Isabella Thomm, Michael Stilkerich, Christian Wawersich, Wolfgang Schröder-Preikschat. KESO An Open-Source Multi-JVM for Deeply Embedded Systems, Friedrich-Alexander University Erlangen-Nuremberg, Germany, 2010.
4. KESO. URL: http://www4.cs.fau.de/Research/KESO/ .
5. Excelsior JET. URL: http://www.excelsiorjet.com/embedded/ .
6. http://www.embeddedinsights.com/channels/topics/ide/ .
7. MPLABX. URL: http://www.microchip.com/pagehandler/en-us/family/mplabx/ .
8. CodeWarrior.URL: http://www.freescale.com/webapp/sps/site/homepage.jsp?code=CW_HOME .
9. Atmel Studio. URL: http://www.atmel.com/tools/atmelstudio.aspx?tab=overview .
10. CCStudio. URL: http://www.ti.com/tool/ccstudio .
11. Amulette Technologies. URL: http://www.amulettechnologies.com/products/gui-design-software .
12. Ankush Varma and Shuvra S. Bhattacharyya. Java-through-C Compilation: An Enabling Technology for Java in Embedded Systems, Febraury 2004 .
13. Sable Publications. URL: http://www.sable.mcgill.ca/publications/thesis/#michaelsMastersThesis .
14. ICELAB. URL: http://www.icelab.dk/ .
15. IAR. URL: www.iar.com .
16. Stephan Korsholm, Kasper Søe Luckow, Bent Thomsen. Towards a Real-Time, WCET Analysable JVM Running in 256kB of Flash Memory, 2011 .
17. Martin Schoeberl, Christian Thalinger, Stephan Korsholm, Anders P. Ravn. Hardware Objects for Java. URL: http://www.jopdesign.com/doc/hwobj.pdf .
18. Michael Stilkerich, Isabella Thomm, Christian Wawersich, Wolfgang Schröder-Preikschat. Tailor-made JVMs for statically configured embedded systems, Concurrency and Computation: Practice & Experience, June 2012, Pages 789-812 .
19. ICELAB. URL: http://www.icelab.dk/evaluation.pdf .
20. Per Bothner. Compiling Java for Embedded Systems.
21. Stephan E. Korsholm. Java for Cost Effective Embedded Real-Time Software, Ph.D. Dissertation, August 2012.

22. A. Krall and R. Gra. Cacao - a 64 bit javavm just-in-time compiler, 1997.

23. JamVM. URL: http://jamvm.sourceforge.net/ .

24. Jamaica. URL: http://www.aicas.com/jamaica.html .

25. F. Pizlo, L. Ziarek, and J. Vitek. Real time Java on resource-constrained platforms with Fiji vm, 2009.

26. Régis Latawiec. Build small footprint GUIs for ARM Cortex-M designs using Java, IS2T. URL: http://www.embedded.com/design/programming-languages-and-tools/4400464./Building-small-footprint-embedded-GUIs-for-ARM-Cortex-M-designs-using-Java .

27. Jonn Walley. Context-senstitive pointer analysis using binary decision diagrams, dissertation, Stanford University, 2005.

28. Kevin Bierhoff. Alias Analysis with bddbddb, 2005

29. Bddbddb. URL: http://bddbddb.sourceforge.net .

30. Mark Marron. Modeling The Heap: A Practical Approach, dissertation, B.A. Mathematics, University of California Berkeley, 2001.

31. Ахо, Альфред В., Лам, Моника С., Сети, Рави, Ульман, Джеффри Д. Компиляторы: принципы, технологии и инструментарий, 2-е изд., изд. Вильямс, 2011.

32. Tal Lev-Ami and Mooly Sagiv. TVLA: A System for Implementing Static Analysis, Tel-Aviv University, 2000.

33. Mool Sagiv and Thomas Reps and Reinhard Wilhelm. Parametric Shape Analysis via 3-Valued Logic, 2002.

34. Pascal Sotin, Bertrand Jeannet, Xavier Rival. Concrete Memory Models for Shape Analysis, 2010.

35. Maria Alpuente, Marco A. Feliu, Cristophe Joubert, Alicia Villanueva. Using Datalog and Boolean Equation Systems for Program Analysis", 2009.

36. Yannis Smaragdakis and Martin Bravenboer. Using Datalog for Fast and Easy Program Analysis, 2009.

37. Doop. URL: http://doop.program-analysis.org/ .

38. M. Lam & J. Whaley. Advanced Compilers, lections, Stanford University.

39. Roman Manevich. Data Structures and Algorithms for Efficient Shape Analysis // School of Computer Science, Tel-Aviv University, Israel, January, 2003.

40. Сафронов А., Намиот Д. Среда разработки программного обеспечения для встраиваемых систем на основе JME //International Journal of Open Information Technologies. – 2013. – Т. 1. – №. 2. – С. 17-24.