

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОНИКИ И МАТЕМАТИКИ
им. А.Н. ТИХОНОВА

Юндин Владислав Андреевич

**РАЗРАБОТКА СИСТЕМЫ КОНТРОЛЯ И УПРАВЛЕНИЯ
ЭНЕРГОПОТРЕБЛЕНИЕМ ЭЛЕМЕНТОВ ГРАФИЧЕСКОГО
ИНТЕРФЕЙСА НА МОБИЛЬНЫХ УСТРОЙСТВАХ**

Выпускная квалификационная работа по направлению подготовки
09.03.01. Информатика и вычислительная техника
студента образовательной программы
«Информатика и вычислительная техника»

Студент

В.А. Юндин

Руководитель

Старший преподаватель ДКИ,
А.Ю. Ролич

Соруководитель

Профессор-исследователь,
к.т.н., доцент, Л.С. Восков

Москва, 2020

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОНИКИ И МАТЕМАТИКИ им. А.Н. ТИХОНОВА

«УТВЕРЖДАЮ»

Академический руководитель
образовательной программы
«Информатика и вычислительная техника»

Ю.И. Гудков

«17» 12 2019 г.

ЗАДАНИЕ
на выпускную квалификационную работу бакалавра

студенту группы БИВ161 Юндину Владиславу Андреевичу

1. Тема работы

Разработка системы контроля и управления энергопотреблением элементов
графического интерфейса на мобильных устройствах

Development of a system for monitoring and energy management of graphical interface
elements on mobile devices

2. Требования к работе

2.1. Общие требования к объекту разработки

Необходимо разработать систему, позволяющую определять состав элементов
графического интерфейса на экране и рекомендовать пути оптимизации
энергопотребления

2.2. Требования к функциональности

- система должна находить иерархию элементов графического интерфейса для каждого экрана
- система должна анализировать иерархию элементов графического интерфейса для каждого экрана
- система должна формировать список рекомендаций по оптимизации энергопотребления для каждого экрана

2.3. Требования к информационной и программной совместимости

- операционная система Android версии 5.0 и выше

-
- использование стандартных для Android элементов графического интерфейса
 - возможность подключения к проекту сторонней библиотеки из файла Android Archive (AAR)
-

3. Содержание работы

3.1. Обзор существующих исследований по проблеме оптимизации энергопотребления

3.2. Обзор инструментов для измерения энергопотребления устройства

3.3. Исследование энергопотребления различных элементов графического интерфейса

3.4. Проектирование подключаемой библиотеки для контроля и управления энергопотреблением элементов графического интерфейса

3.5. Разработка подключаемой библиотеки для контроля и управления энергопотреблением элементов графического интерфейса

3.6. Тестирование системы

4. Сроки выполнения этапов работы

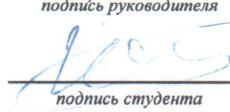
Проект ВКР представляется студентом в срок до:	«10» февраля 2020 г.
Первый вариант ВКР представляется студентом в срок до:	«26» апреля 2020 г.
Итоговый вариант ВКР представляется студентом руководителю до загрузки работы в систему «Антиплагиат» в срок до:	«10» мая 2020 г.

Задание выдано «15» декабря 2019 г.


А.Ю. Ролич

И.О. Фамилия

Задание принято к исполнению «15» декабря 2019 г.


В.А. Юндин

И.О. Фамилия

АННОТАЦИЯ

В настоящее время мобильные устройства пользуются большей популярностью, чем когда-либо прежде. Для улучшения опыта взаимодействия с устройствами необходимо более эффективно использовать их ресурсы, особенно важен вопрос разрядки аккумулятора. Целью данной работы является разработка системы контроля и управления энергопотреблением элементов графического интерфейса на мобильных устройствах с операционной системой Android. В работе представлен способ измерения энергопотребления виджетов с помощью анализа отчётов службы `batterystats`. Процесс измерения управлялся zsh-скриптом через подключение к устройству по USB. Результатом работы стала подключаемая библиотека, способная предложить рекомендации по оптимизации 23 виджетов.

ABSTRACT

Nowadays mobile devices are more popular than ever before. To improve the experience of interaction with devices it is necessary to use their resources more effectively, especially the battery draining issue is important. The purpose of this paper is to develop a system for monitoring and energy management of graphical interface elements on Android devices. This study presents a method for measuring the power consumption of the widget by analyzing the output of batterystats service. The measurement process was controlled by a script written for zsh while the device is connected via USB. The result is a plug-in library that can offer recommendations for optimizing 23 widgets.

Оглавление

Введение	8
1 Актуальность работы	10
2 Обзор литературы	12
2.1 Проблемы пользователей и разработчиков мобильных приложений	12
2.2 Измерение энергопотребления устройств	14
2.3 Оптимизации на уровне исходного кода	15
2.4 Общие методы оптимизации потребления ресурсов для переносных устройств	16
2.5 Энергопотребление в операционной системе Android	18
2.6 Оптимизация энергопотребления экранов мобильных устройств	19
2.7 Выводы	21
3 Инструменты измерения энергопотребления	22
3.1 Способы измерения	22
3.2 Обоснование выбранного инструмента	24
3.3 Анализ системных отчётов	24
3.3.1 Анализ архива отчёта	24
3.3.2 Инструмент dumpsys	25
3.4 Фильтрация данных	28
3.4.1 Служба batterystats	28
3.4.2 Служба cpuinfo	28
4 Процесс сбора данных	29
4.1 Оптимизация сбора данных	29
4.2 Проблемы при сборе данных	30
4.2.1 Сброс статистики	30
4.2.2 Связь с компьютером	30
4.3 Автоматизация процесса	31
5 Составление списка элементов	34
6 Измерение одного виджета	37

6.1	Проектирование автоматических тестов	37
6.1.1	Процесс запуска Activity	37
6.1.2	Передача данных при запуске теста	38
6.1.3	Обработка объекта виджета	39
6.2	Выбор фреймворка автоматического тестирования	41
7	Сравнение результатов измерений в разных условиях	42
7.1	Влияние подключённого USB-кабеля	42
7.2	Влияние запуска через автотест	43
8	Постобработка результатов измерений	46
9	Результаты измерений	47
10	Подключаемая библиотека	49
10.1	Проектирование библиотеки	49
10.2	Язык программирования и инструментальные средства	51
10.3	База данных	52
10.4	Разработка	54
10.4.1	Структура проекта	54
10.4.2	Написание кода библиотеки	54
11	Итоги работы	57
Заключение		58
Глоссарий		61
Перечень сокращений, условных обозначений, символов и терминов		62
Список использованных источников		63
Список сущностей пакета android.widget в Android 10 SDK Platform		66

ВВЕДЕНИЕ

Современная жизнь немыслима без смартфонов и разных мобильных приложений, которые могут неэффективно потреблять ресурсы устройства. Значительные проблемы в оптимизации пользовательского опыта возникают в вопросах разряда батареи. Энергоэффективность приложений — одна из важнейших проблем, с которой сталкиваются как разработчики, так и пользователи [1; 2]. Низкая энергоэффективность приложения ускоряет разрядку смартфона и может даже стать основанием для удаления приложения [3]. Данная проблема имеет популярность среди исследователей, и является предметом большого количества работ. Предложено множество способов снижения энергопотребления, но мне хотелось бы проверить эффективность метода, который основан на замене менее эффективных виджетов на экране более эффективными.

Цель и задачи Конечной целью выпускной квалификационной работы является разработка системы контроля и управления энергопотреблением элементов графического интерфейса на устройствах под управлением операционной системы Android. Для её достижения необходимо решить следующие задачи:

- Проанализировать существующие исследования по оптимизации энергопотребления;
- Определить инструменты для измерения энергопотребления Android-смартфонов;
- Измерить энергопотребление различных элементов пользовательского интерфейса;
- Создать библиотеку для мониторинга и управления энергопотреблением графических элементов пользовательского интерфейса

Практическая значимость Решение данной проблемы в первую очередь представляет интерес для тех, кто занимается разработкой приложений для Android. Любой инженер, которому важно количество потребляемой приложением энергии, сможет встроить библиотеку в приложение для выявления наиболее затратных элементов интерфейса. Также полученная библиотека составит список рекомендаций по снижению нагрузки на аккумулятор устройства. Благодаря

созданной библиотеке, будет увеличено время автономной работы устройств пользователей.

Программные средства Для разработки системы будет использован язык программирования Kotlin и фреймворк автоматического тестирования пользовательского интерфейса Kaspersky. Для измерения энергопотребления использованы отчёты службы batterystats. Для обработки отчётов использованы языки Python и Z shell.

1 Актуальность работы

Смартфон — неотъемлемая часть жизни современного человека. Человек может использовать телефон до 200 раз каждый день [4]. Тем не менее, смартфоны всё ещё не достигли потолка своего развития и многие аспекты нуждаются в улучшении. Одним из таких является энергопотребление устройства.

Проблема заключается в том, что с развитием вычислителей и экранов, которые потребляют всё больше энергии, аккумуляторы должны развиваться с сопоставимой скоростью, чтобы покрыть растущие затраты и оставить время работы устройства от батареи хотя бы на том же уровне. Но развитие аккумуляторов устройств происходит не так быстро [5], что заставляет задуматься о более эффективном расходовании уже имеющейся энергии.

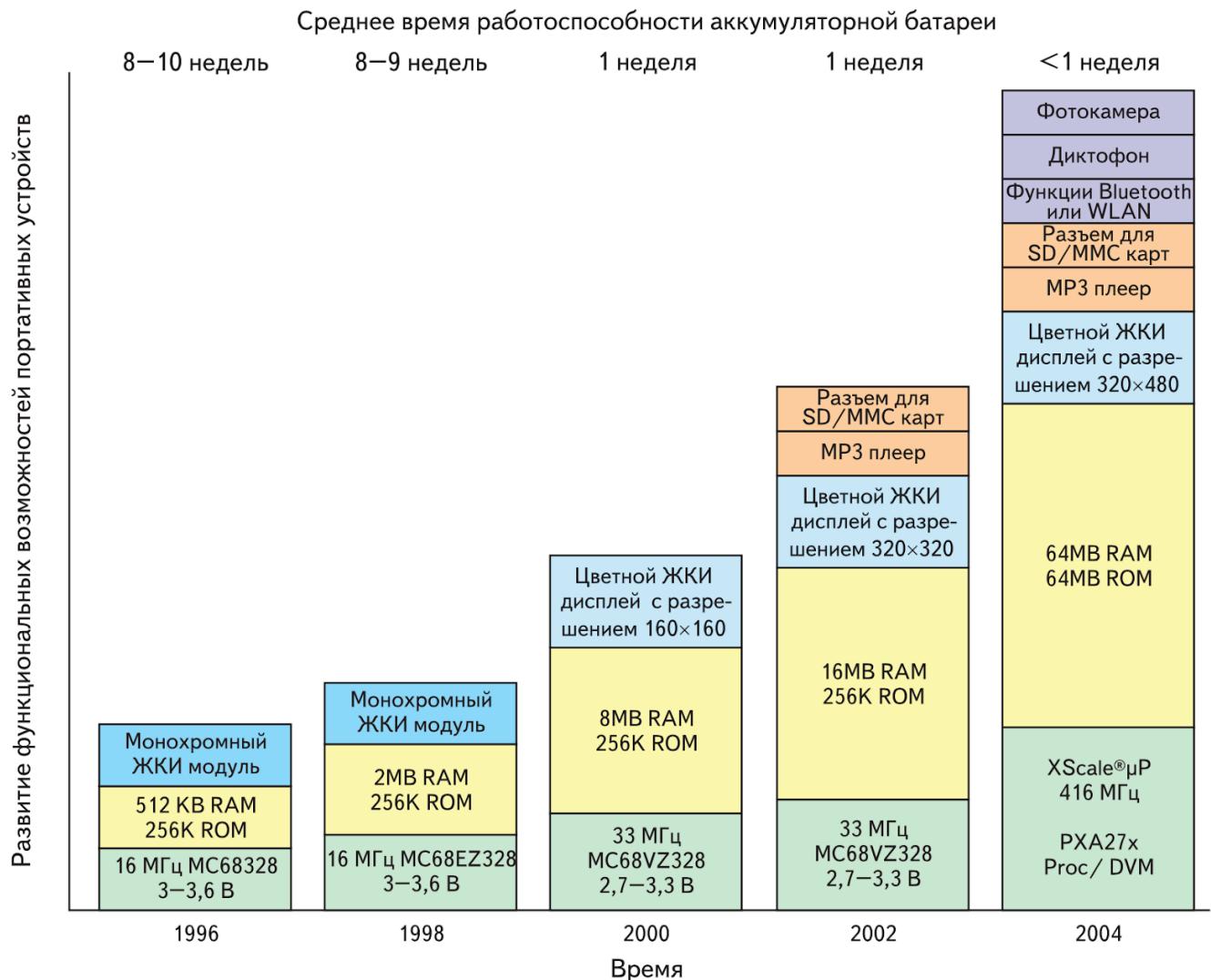


Рис. 1. Среднее время работоспособности аккумуляторной батареи

Раньше мобильные телефоны могли работать более недели от одного заряда

аккумулятора, но с их развитием такая возможность была потеряна. Как видно на диаграмме (рис. 1), процесс увеличения числа возможностей портативных устройств сопряжён с уменьшением времени работы прибора. Если в 1996 году портативное устройство было способно работать 8-10 недель до разрядки аккумулятора, то уже в 2004 году наличие камеры, диктофона, Bluetooth, цветного дисплея и другой функциональности сократили время работы до недели и меньше [6].

Существует большое количество факторов, которые влияют на потребление энергии. Это взаимодействие с сетью, множество сенсоров и датчиков, камера, экран и другие. В академических работах показана возможность оптимизировать потребление многих факторов, например, взаимодействие с интернетом [7] или более оптимальное использование оперативной памяти может продлить срок работы устройства от аккумулятора [8]. Но также исследования показывают, что в большинстве сценариев использования смартфона энергозатраты на экран составляют больше половины всех энергозатрат [9]. Становится очевидным, что именно потребление экрана нуждается в оптимизации в первую очередь.

Имеют место множество подходов к уменьшению потребления экрана, которые основываются на самых разных идеях. Некоторые считают, что изменение цветовой схемы интерфейса на AMOLED экранах поможет снизить затраты [10], другие понижают кадровую частоту и частоту обновления экрана [11; 12].

Новизна моего исследования базируется на предположении, что взаимозаменяемые виджеты со схожей функциональностью потребляют разное количество энергии, что позволяет заменить более затратные виджеты на аналогичные и уменьшить энергопотребление.

Практическое применение моей работы заключается в помощи разработчикам Android-приложений в выборе наиболее эффективных элементов графического интерфейса и улучшение общего качества приложения. Ожидается, что этого удастся достигнуть путём непрерывного сканирования иерархии виджетов с целью поиска тех, которые могут быть заменены на более оптимальные, и сообщения об этом разработчику мобильного приложения.

2 Обзор литературы

В этой главе я рассмотрю работы, затрагивающие проблему энергопотребления мобильных устройств и предлагающие пути её решения. Это необходимо для понимания проделанной исследователями работы, а также изучения методик и инструментов, используемых для проведения измерений.

2.1 Проблемы пользователей и разработчиков мобильных приложений

Я считаю необходимым иметь представление о реальных проблемах мобильных приложений как с точки зрения пользователей, так и с позиции разработчиков приложений. Это поможет правильно расставить приоритеты исследования и разработать наиболее удобное и полезное решение по контролю энергопотребления.

Одна из проблем, освещённая в статье Мана, Гао и др., связана с пользовательским восприятием приложений [1]. Авторами был создан новый фреймворк под названием CrossMiner для автоматического анализа проблем приложений из отзывов пользователей с помощью метода, основанного на ключевых словах. Основываясь на пяти миллионах отзывов пользователей, платформа автоматически фиксирует распределение семи проблем приложения, а именно: “батарея”, “сбой”, “память”, “сеть”, “конфиденциальность”, “спам” и “пользовательский интерфейс”. По итогу было выявлено, что проблемы, связанные со “сбоем” и “сетью”, больше беспокоят пользователей, чем другие проблемы на трех рассматриваемых платформах (Google Play, App Store и Windows Store).

В другой работе задача состояла в том, чтобы определить причины, по которым пользователи выбирают и устанавливают мобильные приложения из магазинов приложений [3]. А также причины, по которым пользователи их удаляют. Было проведено анкетирование с участием 121 респондента из 26 различных стран. Как видно на диаграмме (рис. 2), самыми частыми причинами установки приложения являются его описание, оценки пользователей и скриншоты приложения. Согласно другим данным (рис. 3) в топе причин удаления — бесполезность, сбои, высокое использование оперативной памяти.

С точки зрения инженеров [2] наиболее остро стоят следующий вопросы:

- Улучшение пользовательского опыта;

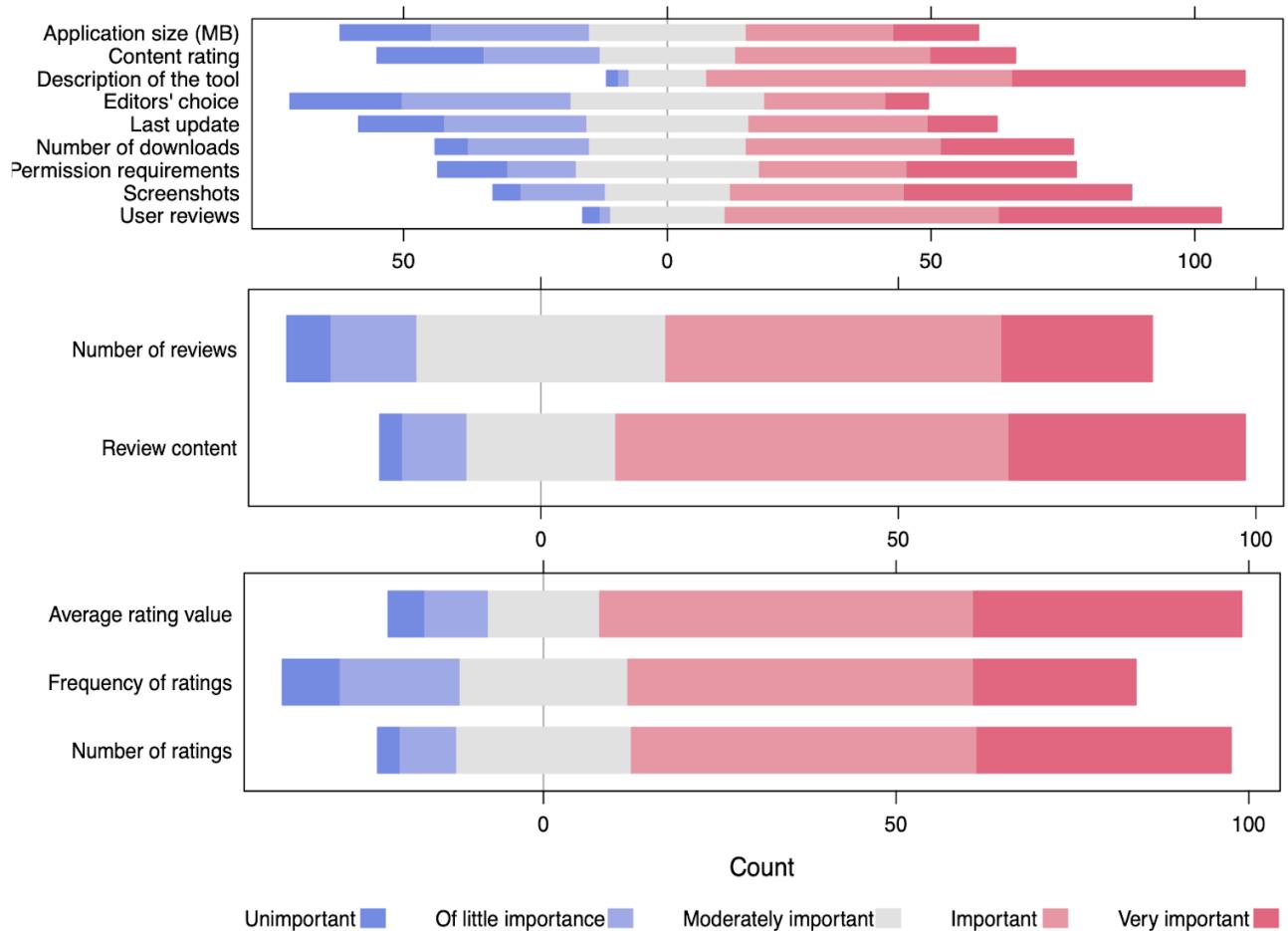


Рис. 2. Причины установки приложений пользователями

- Нефункциональные требования (производительность, энергоэффективность, надёжность, качество, безопасность);
- Процессы, инструменты и архитектура;
- Переносимость на другие платформы.

Эти пункты являются лишь подмножеством возможных тем исследований в области разработки программного обеспечения для мобильных приложений, но служат для обозначения широты исследовательских потребностей и возможностей в этой формирующейся области.

2.2 Измерение энергопотребления устройств

Как мы выяснили ранее, понимание энергопотребления компонентов смартфона является одной из ключевых областей интереса для конечных пользователей, а также разработчиков приложений и системного программного обеспечения. Существующая литература предлагает множество решений для

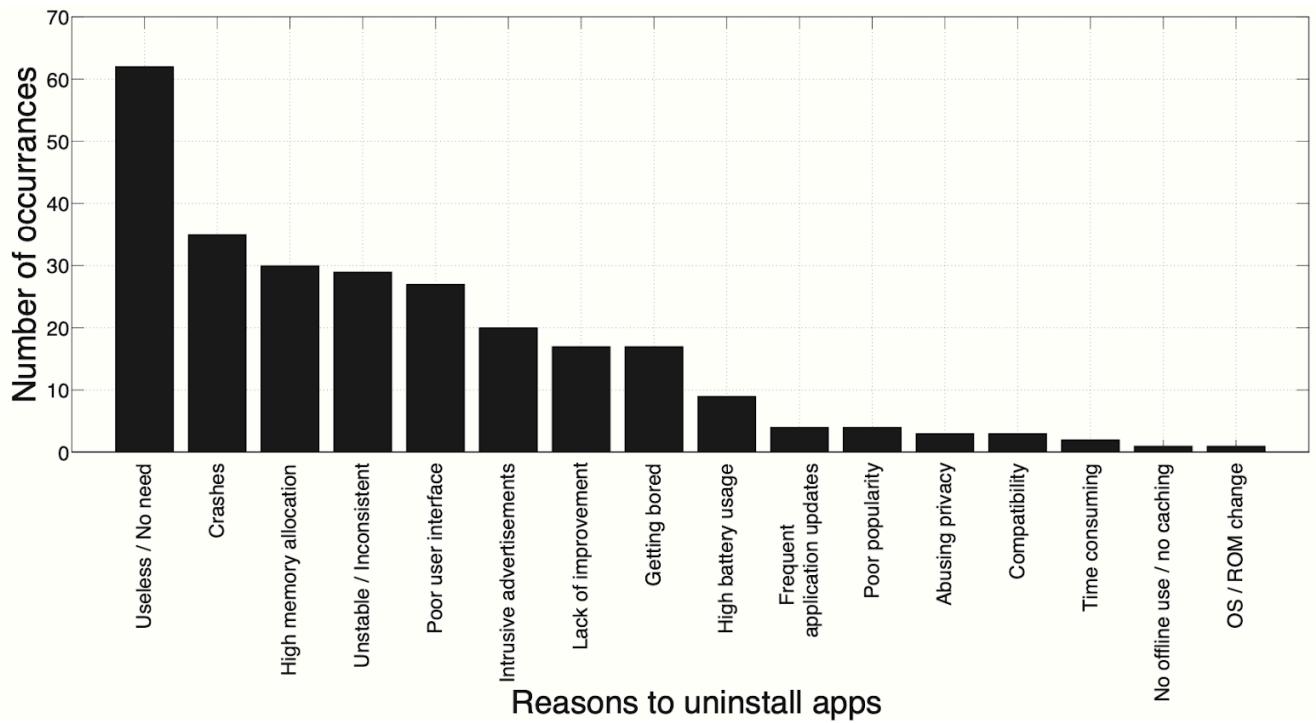


Рис. 3. Причины удаления приложений пользователями

эффективного измерения энергопотребления, что помогает определить влияние различных подходов к оптимизации.

В зарубежных исследованиях нередко предлагаются собственные инструменты измерения. Так, в департаменте компьютерных наук университета Ёнсе в Южной Корее, был разработан AppScope — приложение для автоматического измерения энергопотребления приложений Android с помощью мониторинга активности ядра [13]. Оно отслеживает системные вызовы, а также анализирует данные механизма IPC операционной системы.

В статье Сео, Малека и Медвидовика представлена структура для оценки энергопотребления программных систем на основе Java [14]. Её инфраструктура использует компонентную перспективу, что делает ее подходящей для большого класса современных распределенных, встроенных и распространяющихся приложений. В большом количестве сценариев распределенных приложений платформа показала очень хорошую точность в целом, давая результаты, которые были в пределах 5% от фактического потребления энергии, затраченного при работе приложения.

Проектная группа из университета Джорджа Майсона и Национального института стандартов и технологий США представила систему, которая эффективно учитывает энергопотребление всех основных аппаратных подсистем телефона:

процессора, дисплея, графики, GPS, аудио, микрофона и Wi-Fi [15]. Для этого они использовали доли времени для каждой подсистемы, сообщаемые модулем управления питанием операционной системы. Предложенное решение позволяет работать в режиме реального времени без значительного влияния на энергопотребление подконтрольного устройства, что может помочь разработчикам и исследователям принимать более оптимальные решения для повышения энергоэффективности.

2.3 Оптимизации на уровне исходного кода

Одним из инновационных подходов к снижению потребления энергии устройством является оптимизация затрат процессорного времени, которое требуется для выполнения вычислительных задач. В данной главе будут рассмотрены пути оптимизации исходного кода программы с целью сокращения времени нагруженной работы процессора и скорейшего его перехода в энергосберегающий режим.

Одно из исследований показало, что JavaScript экономит больше энергии и работает медленнее, чем другие подходы, и что гибридизация приложений может быть решением для оптимизации приложений как с точки зрения производительности, так и энергопотребления [16]. Среди двух вариантов гибридизации использование NDK является наиболее безопасным вариантом для повышения производительности, но использование веб-подхода может дать ощутимый результат при небольшом количестве кросс-языковых вызовов.

Ещё одно исследование относительно Java сфокусировано вокруг одного из механизмов для снижения требований к памяти, а именно сжатия [17]. В статье рассматривается влияние сжатия на объём использованных системных ресурсов виртуальной машиной Java (JVM). Также обращается внимание на алгоритмы, применимые для этих целей. Опыты показали, что экономия энергии составляет в среднем 21%. В тех приложениях, где декомпрессия играет большую роль, результаты заметно хуже.

2.4 Общие методы оптимизации потребления ресурсов для портативных устройств

Многие проблемы энергопотребления приходилось решать ранее для оптимизации различных портативных устройств. Разработанные методы можно

применить и для улучшения энергоэффективности смартфонов.

В статье Маурицио предлагается значительно пересмотреть подход к производству устройств с учётом интересов конечного пользователя [18]. А именно: изменить подбор архитектуры и уделять внимание экономии энергии всех стадиях разработки продукта. При выборе архитектуры автор предлагает учитывать критерий энергопотребления, например, энергозатратные архитектуры, использующие ТПЛ и дифференциальные сигнальные методы со скоростью передачи данных более 10 Гбит/с, необходимо сопоставлять с менее энергоёмкими методиками, например, КМОП или использование несимметричных архитектур. В дополнение к данному подходу делается акцент на необходимости дальнейшей оптимизации уже существующих архитектур, дополняя их новыми характеристиками.

Анализ энергопотребления и средней задержки для режима энергосбережения с несколькими циклами ожидания показывает, что можно оптимизировать параметры режимов ожидания с учётом ограничения средней начальной задержки [19]. Предложенная автором методика базируется на математической модели для входного потока, которая применима и для сложных систем с большим числом состояний. Однако методика может быть использована лишь для систем, функционирование которых может быть разбито на циклы регенерации.

Также можно рассмотреть более частные случаи. Например, способы по оптимизации потребления энергии устройствами, функционирование которых управляет микроконтроллером. Они нашли себе применение в том числе и в смартфонах. Обзор охватывает программные, архитектурные и схемотехнические способы снижения потребления [20]:

- Программные способы направлены на снижение вычислительной нагрузки микропроцессора. Приведены такие способы, как использование специфичной математики, так как операции с дробными числами занимают продолжительное время. Также учёт разрядности процессора может сыграть существенную роль в снижении энергопотребления. Ещё один способ предполагает переписывание наиболее ресурсозатратных участков кода на язык ассемблера, но это лишит возможности переноса получившихся решений на другую платформу;
- Архитектурные способы применимы разработчиками микроконтроллера и к ним относится реализация различных режимов энергосбережения, что

позволяет отключать неиспользуемую периферию, а также снижать тактовую частоту процессора. Также отключение неиспользуемых узлов кристалла позволит в разы снизить потребление энергии;

- Схемотехнические методы оптимизации применимы к самой аккумуляторной батарее. Предлагается использование литий-ионных или литий-полимерных источников питания для обеспечения прямого питания схемы от батареи. Также предлагается различное напряжение питания для вычислительного ядра и периферии, но это приводит к снижении тактовой частоты микросхем.

2.5 Энергопотребление в операционной системе Android

Следующая предметная область, которая стоит отдельного упоминания в контексте данной выпускной квалификационной работы, — энергопотребление в операционной системе Android, так как именно для оптимизации работы этой операционной системы будут использованы результаты работы.

В статье Ли и Халфонда была проведена эмпирическая оценка общепринятых методов энергосбережения и повышения производительности [8]. Она позволила определить, в какой степени эти методы смогли сэкономить энергию по сравнению с неоптимизированными аналогами кода. В частности, было обнаружено, что объединение сетевых пакетов до определённого размера и использование определённых методов кодирования для считывания информации о длине массива, доступа к полям классов и выполнения вызовов приводят к снижению энергопотребления. Однако другие методы, такие как ограничение использования памяти, оказали минимальное влияние на количество затраченной энергии. Эти результаты позволяют разработчикам избежать использования неэффективных подходов.

Работа Ли, Хао и др. по сбору информации о поведении приложений в контексте энергопотребления выявила, что в среднем приложения тратят 60% своей энергии в неактивных состояниях. При этом самым энергоёмким составляющим является взаимодействие устройства с сетью [21]. Также они проанализировали три распространённых метода, используемых в исследованиях, связанных с измерением энергопотребления: использование времени работы для расчёта приближенного значения; использование измерения на уровне миллисекунд; и пренебрежение затратами энергией в состоянии покоя. Существенный недостаток этих методов заключается в большой погрешности измерения, что может исказить результаты

исследований.

2.6 Оптимизация энергопотребления экранов мобильных устройств

Целевым компонентом результата данной работы — системы для оптимизации энергопотребления элементов графического интерфейса — является экран мобильного устройства. В связи с чем стоит рассмотреть ранее изученные подходы к энергооптимизации данного компонента.

Исследования показывают, что потребление может быть уменьшено за счет снижения частоты кадров [11], оптимизации сети [7], использования памяти [8], использования тёмного фона жидкокристаллических экранах [22] и так далее. Эмпирическое исследование показывает, что смартфон потребляет большое количество энергии при отображении интерфейса для пользователя [21].

Ван и Джин [10] описали методику обнаружения энергоёмких интерфейсов и их преобразования для экономии энергии. Их результаты оказали значительное влияние на оценку энергопотребления экрана и, кроме того, их идея была разработана в исследовании Линареса-Васкеса и др. [23]. Суть данного подхода основана на оптимизации цветовой палитры при сохранении допустимого уровня контрастности. Частота обновления дисплея также была изучена научным сообществом. Хуан и др. [12] и Ким и Юнг [24] предполагают, что частота обновления является ключевым фактором, за счёт снижения которого достигается экономия энергии.

Ключевой особенностью исследования Вана и Джина [10] является инструмент, который может анализировать содержимое скриншота. Это модель потребляемой дисплеем мощности, которая предсказывает количество энергии, потребляемой каждым пиксели экрана, принимая во внимание тип экрана и цвет определённого пикселя. Позже оригинальный скриншот трансформируется, чтобы достичь более энергоэффективного пользовательского интерфейса. Преобразованное изображение аналогично анализируется перед сравнением результатов с исходными данными. Результаты работы учёных показывают, что предполагаемая экономия энергии достигает 50% от первоначального значения. Это указывает на то, что их идеи служат хорошей основой для дальнейшего изучения. В данной работе оценка погрешности модели была измерена только на 4 различных устройствах, которые могут не отражать полноту всей картины.

В работе Линарес-Васкес и др. [23] основой подхода является многоцелевое рассмотрение контента. Это позволяет разработчикам достичь компромисса между тремя выделенными целями: снижение энергопотребления на OLED-дисплеях, увеличение контраста между соседними элементами пользовательского интерфейса и сохранение согласованности в использовании цветов по сравнению с оригинальным дизайном. Для достижения такого поведения необходимо найти состояние оптимальности по Парето, в котором улучшение одной из целевых функций не может быть достигнуто без ухудшения других целей. С помощью указанных методов учёным удалось в некоторых случаях снизить энергопотребление на 79%. Этот результат представлен с множеством графически выраженных данных, которые иллюстрируют весь процесс. Вопрос, который остаётся нераскрытым — это сопоставимость опыта пользователя до и после оптимизации питания.

Хуан и др. [12], в свою очередь, демонстрируют взаимосвязь между энергопотреблением и частотой обновления дисплея. В настоящее время частота обновления ЖК-экранов постоянна, что приводит к тому, что экрану приходится рисовать те же кадры без особой на то необходимости. Учёные представили механизм, сокращающий избыточные обновления кадров и доступ к памяти на основании информации о кадровых буферов. Тем не менее, из исследования невозможно сделать вывод об устройствах с типом экрана, отличным от LCD.

В дополнение к вышеупомянутым исследованиям Ким и Юнг [24] придерживались идеи интеллектуальной частоты обновления дисплея. Ключевой особенностью статьи является показатель скорости контента и его отношение к частоте обновления экрана. Предлагаемая метрика скорости контента начинается с определения частоты обновления контента для каждого приложения и затем управления частотой обновления, чтобы оптимизировать энергопотребление без ущерба для пользовательского восприятия. Измерение частоты обновления контента основано на частоте кадров и сравнении кадровых буферов разных кадров для вычитания избыточной частоты кадров. Кроме того, была применена технология ускорения касанием: частота обновления резко возрастает, когда происходит событие касания. Это исследование все ещё имеет некоторые ограничения: только одна модель устройства была протестирована, а типы дисплеев, для которых исследование является действительным, не описаны.

2.7 Выводы

Проведённый обзор литературы показал, что на текущий момент проблема энергопотребления активно разрабатывается исследовательским сообществом. Авторы показали необходимость оптимизации ПО, разработки инновационных программных и аппаратных решений, а также учёта пользовательского восприятия. Тем не менее, остаётся открытым вопрос дальнейшей проработки решений, которые помогут повысить энергоэффективность дисплея как самого энергозатратного компонента смартфона.

3 Инструменты измерения энергопотребления

Измерение энергопотребления приложения в конкретный момент или небольшой промежуток времени — одна из самых важных и сложных задач, которую необходимо решить в ходе выполнения выпускной квалификационной работы.

Необходимость в проведении таких измерений связана с необходимостью формирования базы данных элементов пользовательского интерфейса и советов для их оптимизации. В ходе работы системы контроля и управления энергопотребления сформированная база элементов пользовательского интерфейса будет использоваться для сравнения данных по энергопотреблению текущих элементов с подобными по выполняемой функциональности и поиска наиболее энергоэффективных альтернатив.

3.1 Способы измерения

Для получения энергопотребления конкретного приложения могут быть использованы следующие подходы:

- анализ напряжения на аккумуляторе, получаемого через компонент приложения BroadcastReceiver;
- получение данных по энергопотреблению с помощью инструмента Android Profiler;
- анализ данных по конкретному приложению, собираемых операционной системой, с помощью инструмента Battery Historian;
- анализ напряжения на аккумуляторе, собираемого операционной системой, с помощью Battery Historian;
- измерение энергопотребления всего устройства с помощью стороннего оборудования.

Анализ напряжения, получаемого через BroadcastReceiver страдает от несовершенности API. Разные устройства могут иметь различное поведение относительно частоты обновления данных, но в среднем новое значение рассыпается при уменьшении заряда аккумулятора на один процент. То есть

данний подход позволяет получить лишь около 100 значений за время полного разряда устройства, чего недостаточно для анализа.

Android Profiler и Battery Historian позволяют фильтровать потребление по отдельному приложению, что несомненно является преимуществом. Но Android Profiler не учитывает потребляемую экраном энергию и показывает лишь потребление по четырёхзначной шкале (None, Low, Meduim, High) следующих потребителей: процессор, сеть, геолокация. Этого также не хватит для полноценного анализа.

Battery Historian отображает более точную информацию о потреблении конкретным приложением. Доступна информация по проценту израсходованной батареи, а также процессорное время, использованное приложением. Но доступны только суммарные данные за всё время с последней полной зарядки устройства. То есть нельзя сказать, сколько потребляло приложение в конкретный момент времени. Так как тесты проводятся в рамках одного приложения, использовать Battery Historian без модификаций не получится.

Данные по напряжению в Battery Historian доступны по временным промежуткам, но они довольно хаотичны (рис. 4) и без информации о силе тока, ничего извлечь из них не получится.

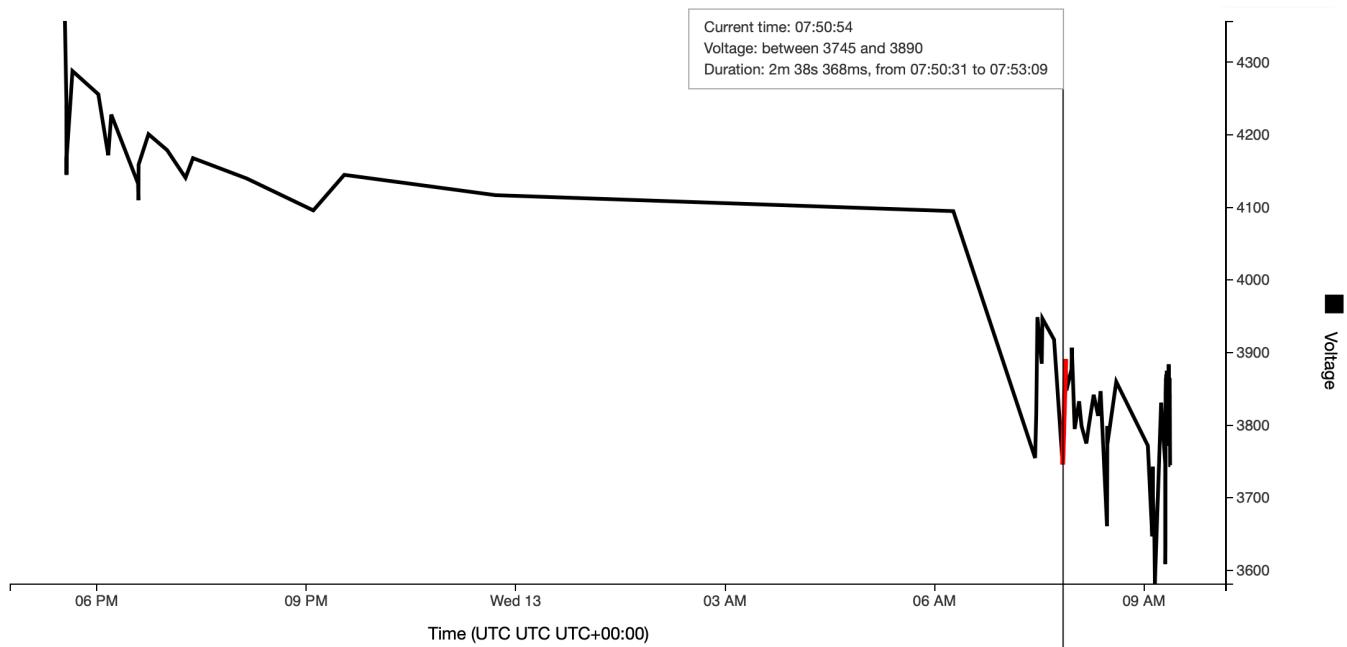


Рис. 4. Напряжение на аккумуляторе устройства

Из-за недостатков остальных подходов, изначальный выбор был сделан в пользу измерения энергопотребления с помощью стороннего оборудования.

3.2 Обоснование выбранного инструмента

Самым надёжным способом в настоящий момент остаётся измерение потребления всего устройства с помощью стороннего оборудования. Данный подход используется практически всеми современными исследованиями, требующими измерения энергопотребления приложения. Устройство для измерения подключается к тестируемому устройству вместо аккумуляторной батареи и записывает показатели потребляемой устройством мощности с некоторым интервалом. Устройства для измерения варьируются от платы Arduino с небольшой программой до оборудования, специально созданного для таких целей. Самым популярным решением является использование Monsoon Power Monitor, именно его я собирался использовать при проведении собственных измерений.

К сожалению, доставка устройства или его аналогов не представляется возможным, в связи с чем, при измерениях придётся данные операционной системы.

Данные ОС выводит в текстовом виде, после чего они могут быть загружены для анализа в Battery Historian. Но так как файлы текстовые, для сокращения времени их анализа, они могут быть проанализированы собственными алгоритмами. К сожалению, найти документации к содержимому файлов не удалось, поэтому формат их содержимого пришлось анализировать самостоятельно.

3.3 Анализ системных отчётов

В Battery Historian загружается zip-архив, получаемый через Android Debug Bridge командой **adb bugreport**. Его следует проанализировать в первую очередь.

3.3.1 Анализ архива отчёта

При распаковке архива мы получаем директорию с файлами, описанными в документации [25]:

- `bugreport-OnePlus3-PKQ1.181203.001-2020-05-13-12-21-11.txt` — самый большой по занимаемому дисковому пространству файл, содержащий всю основную информацию;
- `FS` — директория с некоторыми данными из файловой системы устройства;
- `dumpstate_log.txt` — файл, содержащий информацию о том, как прошёл сбор данных;

- lshal-debug — директория с информацией о слоях аппаратных абстракций операционной системы;
- main_entry.txt — файл, содержащий имя файла с основными данными, в данном случае, первого в списке файла;
- proto — информация от вспомогательных служб в proto-формате;
- version.txt — файл с версией формата отчёта, данном случае используется версия 2.0.

После изучения директории становится понятно, что единственный файл, представляющий интерес — первый в списке. При открытии файла мы видим шапку с информацией о модели устройства, версии ядра операционной системы и так далее (рис. 5). После этого упоминается вызов инструмента **dumpsy**s и представлены разделённые на блоки отчёты различных служб.

```

=====
1 == dumpstate: 2020-05-13 12:21:11
2 =====
3
4 Build: ONEPLUS A3000_28_191104
5 Build fingerprint: 'OnePlus/OnePlus3/OnePlus3T:9/PKQ1.181203.001/1911042108:user/release-keys'
6 Bootloader: unknown
7 Radio: MPSS.TH.2.0.c1.9-00102-M8996FAAAANAZM-1.197095.1.198697.1
8 Network: YOTA
9 Kernel: Linux version 3.18.120-perf+ (OnePlus@ubuntu-15) (gcc version 4.9.x 20150123 (prerelease) (GCC)
   ) #1 SMP PREEMPT Mon Nov 4 21:23:10 CST 2019
10 Command line: sched_enable_hmp=1 sched_enable_power_aware=1 app_setting.use_32bit_app_setting_pro=1 and
    roidboot.hardware=qcom user_debug=31 msm_rtb.filter=0x237 ehci-hcd.park=3 lpm_levels.sleep_disabled=1 c
    ma=32M@0-0xffffffff firmware_class.path=/vendor/firmware_mnt/image loop.max_part=7 buildvariant=user an
    droidboot.bootdevice=624000.ufshc androidboot.verifiedbootstate=green androidboot.veritymode=enforcing
    androidboot.keymaster=1 androidboot.serialno=301a42fd androidboot.authorized_kernel=true androidboot.ba
    seband=msm mdss_mdp.panel=1:dsi:0:qcom,mdss_dsi_samsung_s6e3fa5_1080p_cmd:1:none:cfg:single_dsi fpsimd.
    fpsimd_settings=0 app_setting.use_app_setting=0 androidboot.mode=reboot console=ttyHSL0,115200,n8 andro
    idboot.console=ttyHSL0 earlycon=msm_hsl_uart,0x75B0000 androidboot.project_name=15811 androidboot.rf_v
    ersion=32 androidboot.hw_version=28 ddr_manufacture_info=Samsung ddr_row0_info=16 androidboot.pcba_numb
    er=001581307322032000001629 kmemleak_detect=true androidboot.enable_dm_verity=1 androidboot.secboot=ea
    bled androidboot.angela=disabled androidboot.nobatt=0 androidboot.rpmb_enable=true androidboot.type=norm
    al
11 Uptime: up 9 weeks, 1 day, 1 hour, 9 minutes
12 Bugreport format version: 2.0
13 Dumpstate info: id=2 pid=27566 dry_run=0 args=/system/bin/dumpstate -S -d -z -o /data/user_de/0/com.and
    roid.shell/files/bugreports/bugreport extra_options=
14
15 ----- DUMPSYS CRITICAL (/system/bin/dumpsy) -----
16 -----
17 DUMP OF SERVICE CRITICAL SurfaceFlinger:
@
```

Рис. 5. Шапка отчёта устройства

3.3.2 Инструмент dumpsy

Из найденной документации к **dumpsy**s [26] становится понятно, что отчёты службы можно получать по отдельности, что удобно, так как файл отчёта,

полученный из архива, содержит более 500 тысяч строк и понадобится далеко не вся информация, которую он содержит. В документации можно найти команду **adb shell dumpsys batterystats**, которая возвращает статистику по расходу заряда батареи устройства. Возвращаемый файл достаточно объёмный, наибольший интерес представляет раздел **Statistics since last charge**, так как он содержит информацию по конкретным приложениям.

В подразделе **Estimated power use (mAh)** отображаются затраты, разделённые на Uid (рис. 6). Uid — это пользовательский идентификатор в Unix-подобных операционных системах, он уникален для каждого приложения и по нему можно отслеживать энергопотребление и затраченное процессорное время.

```
Estimated power use (mAh):
Capacity: 3400, Computed drain: 15.2, actual drain: 0
Screen: 13.4 Excluded from smearing
Uid u0a33: 0.464 ( cpu=0.464 ) Including smearing: 0.489 ( proportional=0.0253 )
Idle: 0.379 Excluded from smearing
Uid 1041: 0.376 ( cpu=0.376 ) Excluded from smearing
Uid 0: 0.285 ( cpu=0.285 ) Excluded from smearing
Uid u0a10: 0.114 ( cpu=0.105 sensor=0.00842 ) Including smearing: 0.120 ( proportional=0.00620 )
Uid 1000: 0.0888 ( cpu=0.0888 ) Excluded from smearing
Uid 2000: 0.0137 ( cpu=0.0137 ) Excluded from smearing
Uid 1036: 0.0112 ( cpu=0.0112 ) Excluded from smearing
Uid u0a132: 0.0104 ( cpu=0.0104 ) Including smearing: 0.0110 ( proportional=0.000568 )
Uid u0a662: 0.00858 ( cpu=0.00858 ) Including smearing: 0.00904 ( proportional=0.000468 )
Uid u0a45: 0.00685 ( cpu=0.00685 ) Including smearing: 0.00723 ( proportional=0.000374 )
Uid u0a25: 0.00595 ( cpu=0.00595 ) Excluded from smearing
Uid u0a69: 0.00492 ( cpu=0.00492 ) Including smearing: 0.00519 ( proportional=0.000269 )
Uid u0a22: 0.00408 ( cpu=0.00408 ) Including smearing: 0.00430 ( proportional=0.000223 )
Uid 1001: 0.00119 ( cpu=0.00119 ) Excluded from smearing
Uid 1066: 0.00119 ( cpu=0.00119 ) Excluded from smearing
Uid u0a5: 0.00119 ( cpu=0.00119 ) Excluded from smearing
Uid u0a18: 0.000659 ( cpu=0.000659 ) Including smearing: 0.000695 ( proportional=0.0000360 )
Uid u0a601: 0.000395 ( cpu=0.000395 ) Including smearing: 0.000417 ( proportional=0.0000216 )
Uid u0a666: 0.000395 ( cpu=0.000395 ) Including smearing: 0.000417 ( proportional=0.0000216 )
Uid u0a32: 0.000265 ( cpu=0.000265 ) Including smearing: 0.000279 ( proportional=0.0000144 )
Uid u0a112: 0.000264 ( cpu=0.000264 ) Including smearing: 0.000278 ( proportional=0.0000144 )
```

Рис. 6. Подраздел с затратами на каждый Uid

Также в разделе есть подраздел с общей статистикой для каждого Uid в системе, там не указана израсходованная энергия, но есть информация о процессорном времени, потраченным на это приложение (рис. 7).

Также была использована служба **cputinfo**, предоставляющая дополнительную информацию о затраченном процессорном времени. Обычно сервис предоставляет незначительную информацию, но в отчётах устройства, используемого для тестирования (OnePlus A3000), имелись дополнительные данные, разделённые по времени и Uid, что предоставляет возможности для дополнительного анализа (рис. 8).

```

u0a662:
Wake lock *launch* realtime
Foreground activities: 5m 3s 34ms realtime (0 times) (running)
Top for: 5m 1s 131ms
Cached for: 1s 803ms
Total running: 5m 2s 934ms
Total cpu time: u=590ms s=60ms
Proc com.yundin.estimation:
CPU: 0ms usr + 0ms krn ; 600ms fg
1 starts

```

Рис. 7. Подраздел со статистикой по Uid u0a662

```

$CPU TRACK:v2      uid pid name percent utime stime uptime
[ 2020-05-15 21:54:00.023 to 2020-05-15 21:55:49.730 109708ms 33C]
1000 542 android.hardware.graphics.composer@2.1-service 11 5990 6170 109686
1000 594 surfaceflinger 28 17300 14440 109684
1046 956 media.codec 38 17660 24640 109682
10086 6863 com.google.android.youtube 78 55250 30600 109681
[ 2020-05-15 21:55:49.730 to 2020-05-15 21:57:11.010 81280ms 33C]
1000 594 surfaceflinger 23 10420 8640 81297
1046 956 media.codec 26 8780 12600 81300
10086 6863 com.google.android.youtube 57 29480 17220 81306
[ 2020-05-15 21:57:11.010 to 2020-05-15 21:58:32.349 81339ms 35C]
1000 594 surfaceflinger 22 10220 7790 81323
1046 956 media.codec 21 6930 10190 81322
10086 6863 com.google.android.youtube 45 23790 13020 81314
10646 7738 ru.beru.android 28 18470 4530 81314
[ 2020-05-15 21:58:32.349 to 2020-05-15 22:01:52.273 199924ms 34C]
1000 594 surfaceflinger 28 30600 25820 199946
1046 956 media.codec 35 29620 41910 199947
10086 6863 com.google.android.youtube 75 97670 52440 199956
[ 2020-05-15 22:01:52.273 to 2020-05-15 22:02:11.369 19096ms 34C]
1000 542 android.hardware.graphics.composer@2.1-service 11 1010 1140 19109
1000 594 surfaceflinger 29 3050 2640 19111
1046 956 media.codec 38 3180 4180 19115
10086 6863 com.google.android.youtube 80 10040 5350 19114
[ 2020-05-15 22:02:11.369 to 2020-05-15 22:05:09.829 178460ms 35C]
1000 594 surfaceflinger 28 27480 23520 178439
1046 956 media.codec 38 27580 41180 178434
10086 6863 com.google.android.youtube 78 90210 49400 178427
-----
```

Рис. 8. Дополнительная информация сервиса cpiinfo на устройстве OnePlus A3000

Получаемые с помощью служб **batterystats** и **cpiinfo** данные уже не содержат нерелевантную информацию от других служб операционной системы, но всё ещё имеют объём 3-5 тысяч строк. Понадобится инструмент фильтрации нужной информации.

3.4 Фильтрация данных

На первом этапе фильтрации нужно оставить только информацию, которая относится к тестируемому приложению.

3.4.1 Служба **batterystats**

Для службы **batterystats** это строка с нужным Uid из подраздела **Estimated power use (mAh)**, а также подраздел с общей статистикой по Uid. Для фильтрации этой информации на языке Python был написан скрипт **filter_bat.py**. Скрипт принимает в качестве аргументов командной строки имена файлов, которые ему требуется отфильтровать. Отфильтрованные версии файлов помещаются в директорию **batt_results_filtered/**, которую создаёт скрипт.

3.4.2 Служба **sruinfo**

Из службы **sruinfo** понадобятся такие строки, которые отражают данные для промежутка времени, в которое производилось тестирование, и содержат информацию по Uid тестируемого приложения. Также понадобятся сами строки с датами, так как временные промежутки в файле не равные. Для фильтрации данных на языке Python был написан скрипт **filter_sru.py**. Он так же принимает в качестве аргументов командной строки имена файлов, которые ему требуется отфильтровать, а результат помещает в созданную директорию **sru_results_filtered/**.

4 Процесс сбора данных

Сложность проведения измерений энергопотребления приложения заключается в наличии большого количества факторов, влияющих на показатели энергопотребления, которые не связаны непосредственно с работой приложения. Такими факторами могут быть другие приложения, выполняющие работу в фоновом режиме, задачи операционной системы, а также особенности конкретного устройства.

4.1 Оптимизация сбора данных

Избавиться от влияния всех факторов не представляется возможным, однако для уменьшения их влияния можно предпринять следующие меры:

- проводить тестирование на устройстве без сторонних приложений;
- проводить тестирование на устройстве с операционной системой без сервисов Google Play Services, которые выполняют фоновые задачи операционной системы;
- проводить тестирование всех элементов интерфейса на одном и том же устройстве;
- проводить тестирование при активированном на устройстве режиме полёта;
- проводить тестирование с максимальной яркостью экрана.

К сожалению, доступа к устройству без сторонних приложений и с операционной системой без сервисов Google Play Services, не имеется. Тестирование будет проводится на устройстве OnePlus A3000 с операционной системой Android 9 и версией ядра 3.18.120-perf+. На устройстве будет активирован режим полёта и установлена максимальная яркость экрана. Также будут закрыты все сторонние приложения.

Несмотря на то, что данные меры снижают погрешность при измерениях, они не исключают её полностью. Поэтому будет необходимо провести несколько измерений и дополнительно обработать результаты, чтобы получить усреднённые значения.

4.2 Проблемы при сборе данных

Используемый метод получения данных от операционной системы не лишен недостатков. Стоит их рассмотреть.

4.2.1 Сброс статистики

Как уже упоминалось, служба Battery Historian предоставляет только суммарные данные за время с последней полной зарядки устройства, что мешает получать данные по конкретному виджету. Такое поведение обусловлено тем, что именно в таком формате предоставляет данные служба **batterystats**, анализ которой будет производится и в этой работе.

Для решения данной проблемы была использована команда **adb shell dumpsys batterystats --reset**, которая помогает вручную сбросить все имеющиеся данные и начать запись статистики сначала.

В случае сброса данных перед измерением определённого виджета и сбора статистики после измерения, полученные данные будут отражать статистику по конкретному виджету, что и требуется.

4.2.2 Связь с компьютером

Для сбора результатов измерения виджета и сброса статистики перед измерением следующего может понадобится подключение устройства к компьютеру, который будет вызывать эти команды через Android Debug Bridge.

Подключение может быть осуществлено через USB-кабель, либо через локальную сеть, когда устройства подключены к одной точке доступа Wi-Fi.

В случае общей точки доступа, устройство имеет доступ к интернету, что не позволит ему находиться в режиме полёта и может сильно исказить результаты при обращении других приложений к сети. Данный способ подключения далее не рассматривается.

В случае подключения по кабелю устройство находится в состоянии зарядки. Однако когда устройство заряжается, сбор статистики службой **batterystats** приостанавливается. Для решения этой проблемы была использована возможность искусственно выключать режим зарядки на устройстве командой **adb shell dumpsys battery set usb 0**. После команды устройство продолжает заряжаться, но операционная система ведёт себя так, будто устройство не заряжается. Очевидно, что в этом случае результаты также могут быть неточными, поэтому искажения

данного способа будут дополнительно рассмотрены в подразделе 7.1.

4.3 Автоматизация процесса

Как стало понятно в подразделе 3.3.2, между переключением элементов нужно обращаться к командной оболочке операционной системы устройства, чтобы записывать статистику по протестированному элементу на компьютер и сбрасывать её перед тестированием следующего.

В теории, это может быть сделано двумя способами: обращением к командной оболочке из самого приложения, либо обращением к оболочке через средства Android Debug Bridge с компьютера. При проверке оказалось, что первый вариант невозможен, так как требует особого разрешения операционной системы, которое не выдаётся сторонним приложениям [27]. Поэтому сбор и сброс статистики необходимо осуществлять через Android Debug Bridge.

Механизмов обращения из приложения к компьютеру не предусмотрено. Поэтому сделан вывод о том, что тестом должен управлять скрипт, который исполняется на компьютере, он будет сбрасывать данные, запускать измерения с нужным виджетом и записывать результат.

Был написан zsh-скрипт **estimation.sh** (рис. 9), который в цикле для каждого виджета сбрасывает статистику батареи и запоминает время начала теста. Так как способ запуска измерений пока неизвестен, в коде оставлены пустые строки. После запуска измерений, скрипт ожидает до окончания теста плюс одну секунду, после чего записывает время окончания теста, а также собранную статистику. После выполнения измерений для каждого виджета, записывается статистика процессора.

Запись времени начала и конца необходима, чтобы найти результат тестирования в статистике процессора. Она не сбрасывается вместе со статистикой батареи, поэтому необходимо записать время, в которое проводился тест, в файл со статистикой. Статистика процессора отображается за промежуток в последние несколько суток.

```

1  #!/bin/zsh
2
3 TIME_MS=300000
4 VIEWS_COUNT=27
5
6 main() {
7     adb shell exit || exit 1
8
9     [ -d "results" ] && check_result_dir
10    mkdir "results"
11
12    for i in {0..$((VIEWS_COUNT - 1))} ; do
13        echo "Testing view $i..."
14        adb shell dumpsys batterystats --reset || exit 2
15        echo "Batterystats reset success"
16
17        START=$(date +'%Y-%m-%d %T')
18        echo $START
19        echo "Testing..."
20
21        sleep $($TIME_MS/1000. + 1)
22
23        END=$(date +'%Y-%m-%d %T')
24        echo "Testing completed"
25
26        echo "Dumping..."
27        adb shell dumpsys batterystats --write || exit 6
28        adb shell dumpsys batterystats > "results/battery_$i" || exit 3
29        echo $START > "results/cpu_$i"
30        echo $END >> "results/cpu_$i"
31        #adb shell dumpsys cpuinfo >> "results/cpu_$i" || exit 4
32        echo "Dumpsys complete successfully\n"
33    done
34
35    cpuinfo=$(adb shell dumpsys cpuinfo)
36    for f in results/cpu_*; do
37        echo $cpuinfo >> $f
38    done
39 }

```

Рис. 9. Листинг части скрипта для проведения измерений

5 Составление списка элементов

До начала всех измерений необходимо конкретизировать круг всех элементов интерфейса, с которыми итоговая система будет корректно работать.

Было решено использовать только те виджеты, которые могут быть отображены без взаимодействия с другими виджетами, следовательно, могут быть измерены отдельно от всего остального. В противном случае, было бы необходимо измерять виджеты во всех возможных условиях, а это бы в разы увеличило время проведения тестов.

Для составления списка элементов был взят список всех сущностей пакета android.widget в Android 10 SDK Platform (Приложение). После этого из списка были исключены следующие сущности:

- deprecated классы, так как они не рекомендуются к использованию разработчиками ОС и в будущих версиях могут быть удалены;
- интерфейсы и классы-адаптеры, так как они не являются виджетами;
- абстрактные классы, так как невозможно создать объект такого класса;
- вспомогательные классы, так как они не являются виджетами;
- виджеты, требующие наличие адаптера или презентера для отображения;
- наследники представлений, задачей которых является позиционирование других представлений, добавленных к текущему.

После этого остался список самостоятельных элементов, которые можно независимо тестировать:

- AutoCompleteTextView
- Button
- CalendarView
- CheckBox
- CheckedTextView
- Chronometer

- DatePicker
- EditText
- ImageButton
- ImageSwitcher
- ImageView
- MultiAutoCompleteTextView
- NumberPicker
- ProgressBar
- RadioButton
- RatingBar
- SearchView
- SeekBar
- Space
- Switch
- TextClock
- TextSwitcher
- TextView
- TimePicker
- ToggleButton
- VideoView
- View

6 Измерение одного виджета

Ручное проведение измерений потребует огромного количества времени для включения экранов с различными элементами интерфейса, а также данное переключение будет неточным и может исказить результаты. Поэтому необходимо разработать средства, позволяющие автоматизировать процесс отображения нужного виджета.

Чтобы управлять отображением виджетов на экране, был выбран инструментарий для автоматического тестирования графического интерфейса. Он позволяет имитировать реальное взаимодействие пользователя с устройством. В этой работе пользовательские действия не будут имитированы, но это может понадобится для дальнейших исследований.

6.1 Проектирование автоматических тестов

Автоматические тесты графического интерфейса помогут отображать виджеты на экране по команде, поданной компьютером. Это избавляет от необходимости ручного управления переключением элементов и искажений, которые накладывает ручное переключение.

Однако в подразделе 4.2.2 была обозначена необходимость сравнения результатов измерений в условиях подключения к компьютеру по кабелю и отсутствия такого подключения. Это важно, потому что запуск автоматических тестов без подключения к компьютеру невозможен.

6.1.1 Процесс запуска Activity

Появляется необходимость запускать тестирование не только через автоматические тесты, но и с запуском Activity. Запуск Activity может быть произведён как с компьютера, так и без взаимодействия с ним. Также появляется необходимость сравнения результатов измерений при отображении виджета через запуск автотеста и запуск Activity. Результат такого сравнения представлен в подразделе 7.2.

Для создания максимально близких условий последующего сравнения, процесс создания объекта виджета и его добавления к Activity должны быть одинаковы при отображении виджета через запуск автотеста и запуск Activity.

Для достижения такого поведения, именно Activity на стадии своей инициализации должна иметь информацию о номере виджета, и устанавливать

его на экран. После чего Activity должна ждать столько, сколько нужно для тестирования виджета, и завершаться.

Передать информацию об индексе нужного виджета, а также о времени тестирования можно единственным способом: через объект Intent, с помощью которого происходит запуск Activity. Intent может быть сформирован как через adb, так и при запуске Activity через автотест, что создаёт одинаковые условия запуска Activity.

При создании класса Activity создаётся массив всех виджетов. В методе onCreate Activity получает необходимые данные из объекта Intent, создаёт объект класса нужного виджета, устанавливает его в качестве своего наполнения, а также создаёт отложенную задачу на своё завершение, которую помещает в очередь сообщений главного потока. Последнее действие нужно потому, что метод onCreate всегда вызывается в том же потоке исполнения, в котором происходит отрисовка пользовательского интерфейса, и если заставить поток ждать, виджет не будет отображён на экране.

Теперь для запуска измерений нужного виджета через автотест, необходимо лишь сконфигурировать Intent запуска Activity, запустить её и не завершать поток исполнения автотеста до завершения тестирования. Если поток автотеста завершится сразу после запуска Activity, то завершится и сама Activity. Так как в данном случае не имитируется взаимодействие пользователя с виджетом, поток не выполняет никакой работы, а просто ожидает.

6.1.2 Передача данных при запуске теста

Ещё одна проблема появляется при передаче номера тестируемого виджета автотесту. По умолчанию возможности передавать дополнительные данные из команды запуска теста в код теста нет, но её можно добавить, расширив класс Instrumentation. По умолчанию, в приложении используется класс AndroidJUnitRunner в качестве Instrumentation, поэтому мой класс MyTestRunner будет наследовать и расширять возможности AndroidJUnitRunner (рис. 10).

Было решено передавать в приложение 2 параметра: индекс виджета и время его тестирования. Время будет удобнее определять в единственном месте, этим местом будет zsh-скрипт. После этого останется в файле **build.gradle** сменить testInstrumentationRunner на написанный com.yundin.estimation.MyTestRunner.

```

1 package com.yundin.estimation
2
3 import android.os.Bundle
4 import androidx.test.runner.AndroidJUnitRunner
5
6 class MyTestRunner : AndroidJUnitRunner() {
7
8     var testIndex: Int = 0
9     var testingTime: Int = 0
10
11    override fun onCreate(arguments: Bundle?) {
12        super.onCreate(arguments)
13
14        if (arguments != null) {
15            testIndex = arguments.getString("index")?.toInt() ?: 0
16            testingTime = arguments.getString("time")?.toInt() ?: 0
17        }
18    }

```

Рис. 10. Листинг класса MyTestRunner

6.1.3 Обработка объекта виджета

Некоторые элементы, например TextView или ImageView, требуют наполнения каким-нибудь содержимым, чтобы отображаться на экране. Некоторые элементы требуют менее тривиальных манипуляций перед отображением. Чтобы унифицировать процесс обработки элемента перед и после добавления на экран, было принято решение создать класс ViewWrapper, который будет содержать класс требуемого представления, а также методы beforeAdd и afterAdd, которые будут вызваны до добавления на экран и после добавления на экран соответственно.

Если элементу требуется дополнительные действия, как в примерах выше, методы могут быть переопределены в наследниках класса ViewWrapper. При отображении виджета производятся следующие действия:

1. Создаётся объект класса представления, содержащегося в объекте ViewWrapper;
2. Вызывается метод beforeAdd для созданного объекта;
3. Объект устанавливается в качестве наполнения Activity;
4. Вызывается метод afterAdd для добавленного объекта.

Были написаны наследники класса ViewWrapper, которые будут наполнять виджеты минимальным содержимым (рис. 11). В Activity будет формироваться массив, содержащий объект ViewWrapper для каждого тестируемого виджета.

```
9 open class ViewWrapper(val viewClass: Class<*>) {
1
2     open fun beforeAdd(view: View) {}
3     open fun afterAdd(view: View) {}
4 }
5
6 open class TextViewWrapper(className: Class<*>): ViewWrapper(className) {
7
8     override fun beforeAdd(view: View) {
9         (view as TextView).text = "Sample text"
10    }
11 }
12
13 class ImageViewWrapper(className: Class<*>): ViewWrapper(className) {
14
15     override fun beforeAdd(view: View) {
16         (view as ImageView).setImageResource(android.R.drawable.ic_input_add)
17     }
18 }
19
20 class CheckedTextViewWrapper(private val setCheckMark: Boolean = false): TextViewWrapper(CheckedTextView::class.java) {
21
22     override fun beforeAdd(view: View) {
23         super.beforeAdd(view)
24         if (setCheckMark) {
25             view as CheckedTextView
26             view.setCheckMarkDrawable(android.R.drawable.checkbox_on_background)
27             view.setOnClickListener {
28                 view.toggle()
29                 if (view.isChecked) {
30                     view.setCheckMarkDrawable(android.R.drawable.checkbox_off_background)
31                 } else {
32                     view.setCheckMarkDrawable(android.R.drawable.checkbox_on_background)
33                 }
34             }
35         }
36     }
37 }
38
39 class NumberPickerWrapper: ViewWrapper(NumberPicker::class.java) {
40
41     override fun beforeAdd(view: View) {
42         (view as NumberPicker).maxValue = 10
43     }
44 }
```

Рис. 11. Листинг класса ViewWrapper и части его наследников

6.2 Выбор фреймворка автоматического тестирования

Далее необходимо выбрать конкретный инструмент, позволяющий писать автотесты для Android, рассмотрим варианты:

- appium;
- espresso;
- kakao;
- kaspresso.

Appium не является подходящим, так как он не всегда стабильно работает, а также не поддерживает старые версии Android.

У Espresso подобные проблемы отсутствуют, но присутствует проблема с избыточностью синтаксиса и нечитаемости итогового кода.

Kakao и Kaspresso в свою очередь основываются на Espresso, но каждый по-своему исправляет избыточность синтаксиса с помощью возможностей языка Kotlin, но Kaspresso имеет расширенную функциональность, поэтому был выбран данный фреймворк.

7 Сравнение результатов измерений в разных условиях

В подразделах 4.2.2 и 6.1.1 обосновывается необходимость сравнения результатов измерений, проведённых в разных условиях.

7.1 Влияние подключённого USB-кабеля

В этом подразделе будут протестированы сценарии измерений при подключённом USB-кабеле с искусственно отключённым режимом зарядки и при питании только от батареи.

Условия тестирования будут максимально похожими, они описаны в подразделе 4.1. Единственное отличие, помимо подключения к зарядке, будет в способе запуска Activity.

В случае подключения к компьютеру запуск будет происходить также, как при реальных измерениях, с помощью команды **adb shell am start**. В случае отсутствия связи с компьютером, приложение будет запускаться через нажатие на его иконку в списке приложений.

При открытии приложения через нажатие на иконку, будет сформирован Intent без информации о том, какой виджет и на какое время нужно отобразить. Поэтому при отсутствии этих данных в Intent Activity будет использовать значения по умолчанию, они же будут переданы при запуске через adb.

Тестирование будет производится с виджетом, имеющим номер ноль, а именно AutoCompleteTextView. Перед тестом статистика будет сброшена, после теста статистика будет сохранена в файл. Каждый сценарий повторяется трижды, чтобы увидеть искажения от подключения к сети, а также разницу между измерениями, проведёнными по одному сценарию.

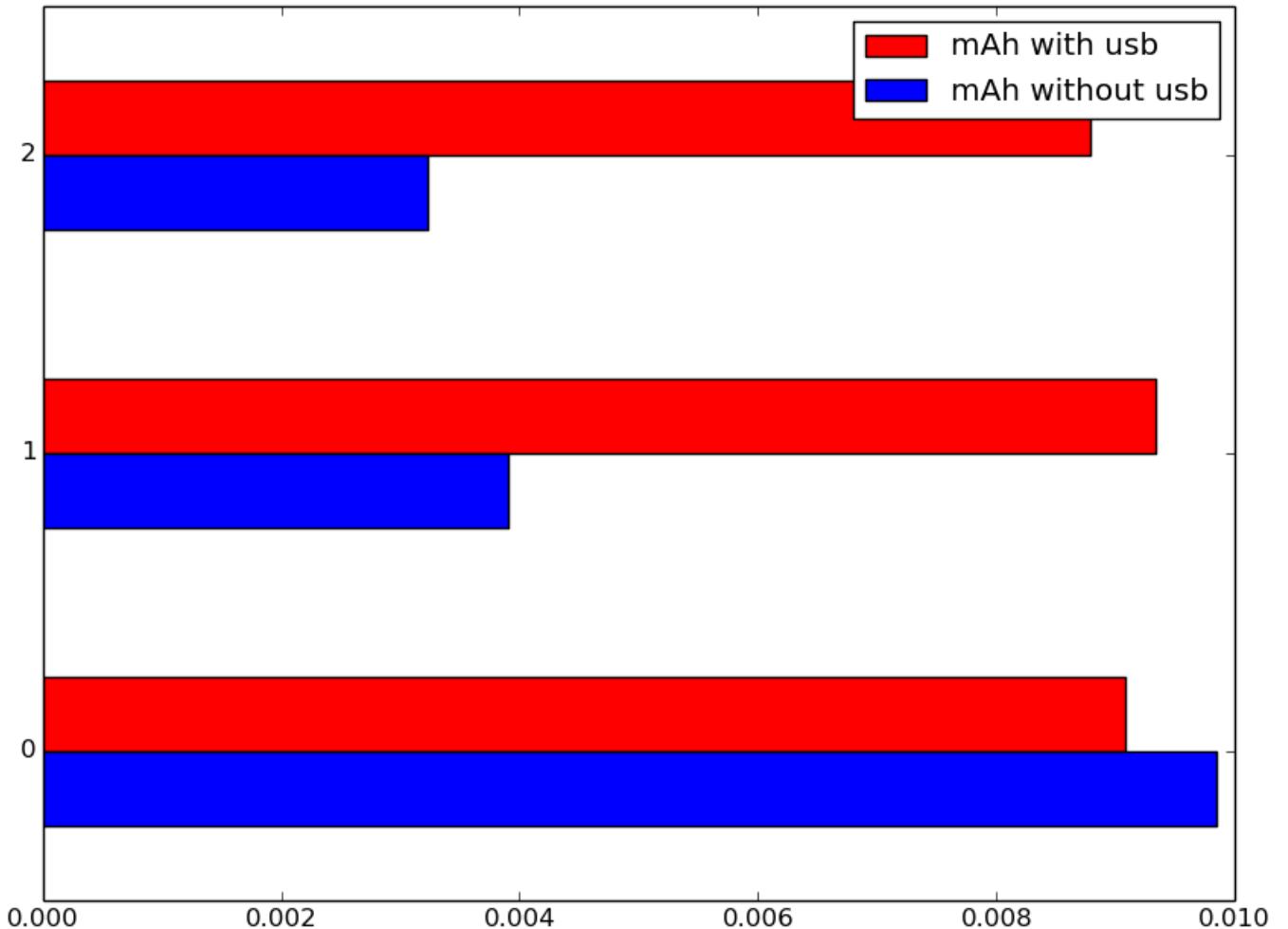


Рис. 12. Результаты сравнения сценариев измерения. Красные столбцы — 3 измерения с подключённым кабелем. Синие столбцы — к устройству ничего не подключено.

Сравнение показало, что результаты измерения по разным сценариям достаточно сильно отличаются (рис. 12), но разница между измерениями без подключения к компьютеру также довольно большая и такому способу доверять сложно. В то же время при подключённом кабеле результаты относительно стабильные.

7.2 Влияние запуска через автотест

Так как в данном случае не будет имитироваться взаимодействие с пользователем, проводить измерения с помощью автотестов необязательно, можно напрямую запускать Activity, передавая нужные параметры. Необходимо сравнить между собой сценарий измерения с использованием автотестов и с запуском Activity напрямую.

Тестирование так же будет производится на виджете AutoCompleteTextView.

Автотест будет запущен командой **adb shell am instrument**. Activity, как и ранее, запускается командой **adb shell am start**.

Стоит отметить, что с точки зрения Activity эти сценарии совершенно ничем не отличаются. Она запускается одинаковым образом и совершает одинаковые действия.

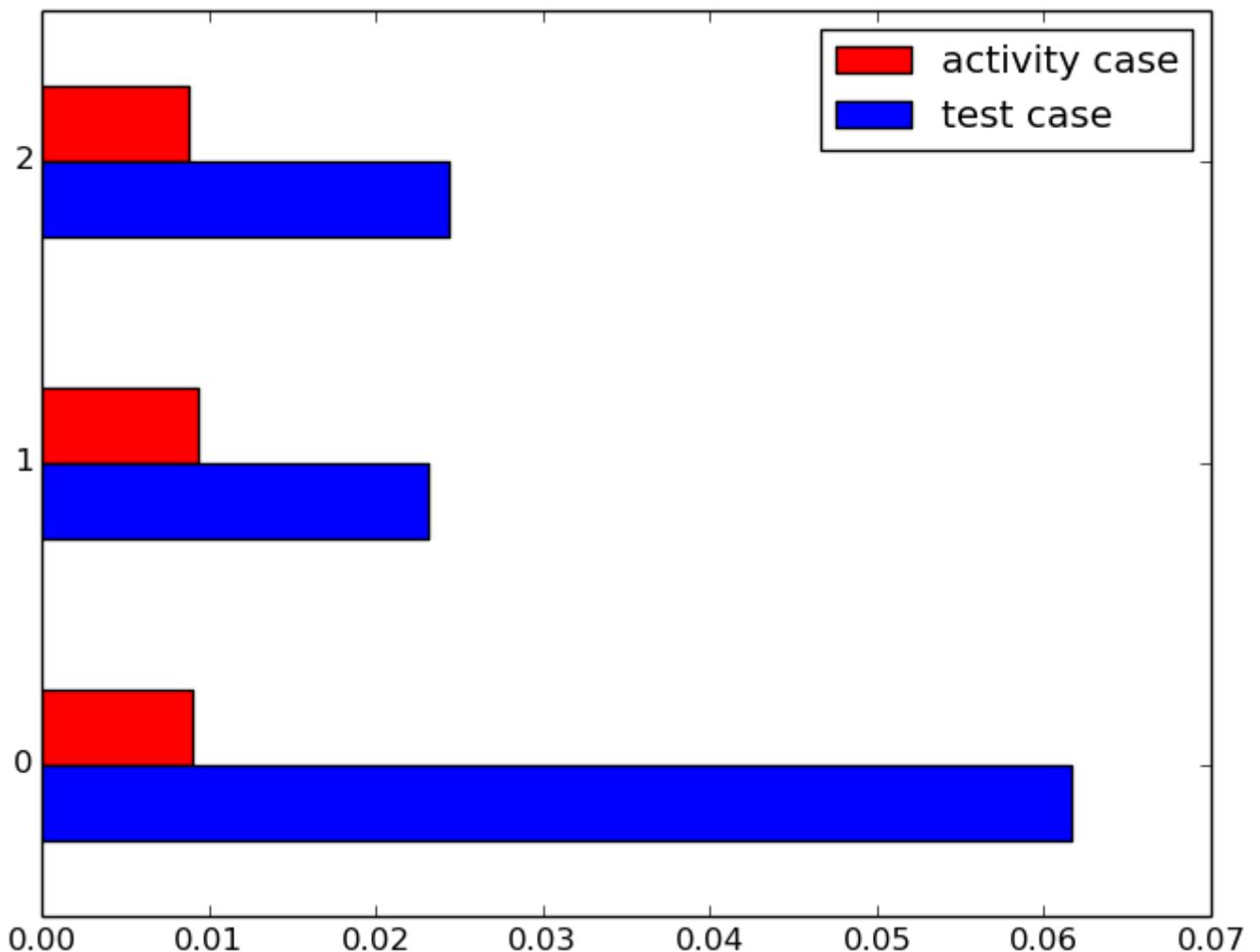


Рис. 13. Результаты сравнения сценариев измерения. Красные столбцы — измерения, запущенные через Activity. Синие столбцы — измерения, запущенные через автотест.

Результаты представлены на рис. 13. Видно, что измерения, запущенные через автотесты, тратят больше энергии, что можно объяснить издержками на дополнительный поток и выполнение дополнительного кода, запускающего процесс тестирования. Также между результатами трёх одинаковых измерений, запущенных через автотесты, есть существенная разница. Причиной этому могут быть индивидуальные особенности используемого устройства или установленной операционной системы.

Сравнение сценариев измерения показало, что стабильней всего себя показывает вызов Activity напрямую при подключённом USB-кабеле. Для реализации данного сценария для всех виджетов скрипт **estimation.sh** был модифицирован до скрипта **activity_estimation.sh** (рис. 14).

```
12  for i in {0..$((VIEWS_COUNT - 1))} ; do
1  echo "Testing view $i..."
2  adb shell dumpsys batterystats --reset || exit 2
3  echo "Batterystats reset success"
4
5  START=$(date +'%Y-%m-%d %T')
6  echo $START
7  echo "Testing..."
8  adb shell am start -n "com.yundin.estimation/com.yundin.estimation.MainActivity" \
9      --ei index $i \
10     --ei delay $TIME_MS
11 #sleep $(( $TIME_MS / 1000. + 1 ))
12 secs=$(( $TIME_MS / 1000 + 1 ))
13 while [ $secs -gt 0 ]; do
14     echo -ne "$secs\033[0K\r"
15     sleep 1
16     : $((secs--))
17 done
18
19 END=$(date +'%Y-%m-%d %T')
20 echo "Testing completed"
21
22 echo "Dumping..."
23 adb shell dumpsys batterystats --write || exit 6
24 adb shell dumpsys batterystats > "results/battery_$i" || exit 3
25 echo $START > "results/cpu_$i"
26 echo $END >> "results/cpu_$i"
27 #adb shell dumpsys cpuinfo >> "results/cpu_$i" || exit 4
28 echo "Dumpsys complete successfully\n"
29 done
30
31 cpuinfo=$(adb shell dumpsys cpuinfo)
32 for f in results/cpu_*; do
33     echo $cpuinfo >> $f
34 done
35 }
```

Рис. 14. Листинг части скрипта activity_estimation.sh

8 Постобработка результатов измерений

В ходе сравнения результатов измерений в разных условиях было обнаружено, что тестируемое приложение не отображается в логах службы `cpiinfo`. Это может быть связано с низкими значениями затрат процессорного времени. Информация этой службы в будущем использована не будет.

После работы скрипта `filter.bat.py` файлы содержат только информацию по тестируемому приложению, но она понадобится не в полном объёме. Для сравнения достаточно получить затраченное процессорное время, а также количество mAh.

Скрипт `concat_filtered.sh` исключает из ранее отфильтрованных данных все подробности, кроме описанных выше. Также происходит сбор данных для всех виджетов в один файл для удобства дальнейшей обработки.

Скрипт `sum_cpi.py` помогает просуммировать процессорное время, потраченное на исполнение системного и пользовательского кода во время работы приложения. Изначально данные представлены в формате, понятном человеку (например, `u=240ms s=50ms`), но компьютер сравнивать такие строки не сможет. Скрипт переводит данную строку в количество миллисекунд и записывает следующей строкой.

Скрипт `median.py` подставляет реальные имена виджетов вместо их номеров и считает медианное значение всех измерений. Используется именно медианное значение, так как из тестов сценариев понятно, что могут наблюдаться выбросы, сильно искажающие среднее значение.

И наконец, скрипт `chart.py` сортирует результаты разных виджетов по затраченному процессорному времени и отображает в виде диаграммы.

9 Результаты измерений

Время измерения каждого виджета было установлено на 5 минут. Увеличение времени измерения одного виджета привело бы к существенному увеличению времени всего тестирования. Было произведено 3 измерения для каждого виджета, чтобы была возможность усреднить результаты.

После проведения всех измерений и обработки результатов получилась диаграмма, представленная на рис. 15 и 16.

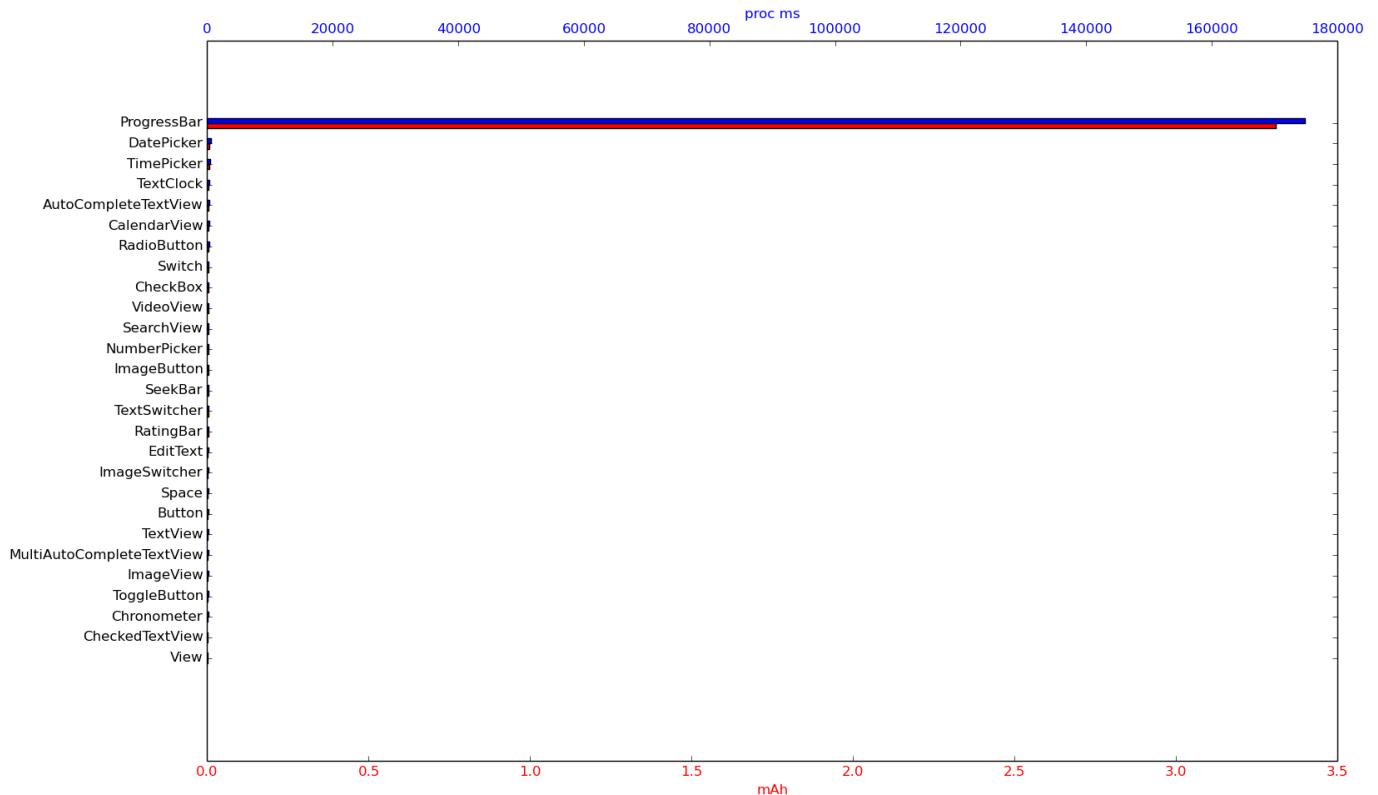


Рис. 15. Результаты измерений. Синие столбцы — процессорное время в мс. Красные столбцы — количество затраченных мАh.

Видно, что виджет ProgressBar тратит энергии в десятки раз больше остальных виджетов. Это неудивительно, потому что он проигрывает непрекращающуюся анимацию, которая требует постоянных расчётов и обновления содержимого экрана. Также в движении находится виджет VideoView, который воспроизводит видеофрагмент, но показатели его потребления не выбиваются из ряда остальных.

Из результатов можно сделать и более неожиданные выводы. Например, виджет Space, заявленный как легковесное View, на деле потребляет больше ресурсов устройства, чем оригинальное View. Также MultiAutoCompleteTextView потребляет ощутимо меньше энергии, чем подобные ему AutoCompleteTextView, и даже EditText.

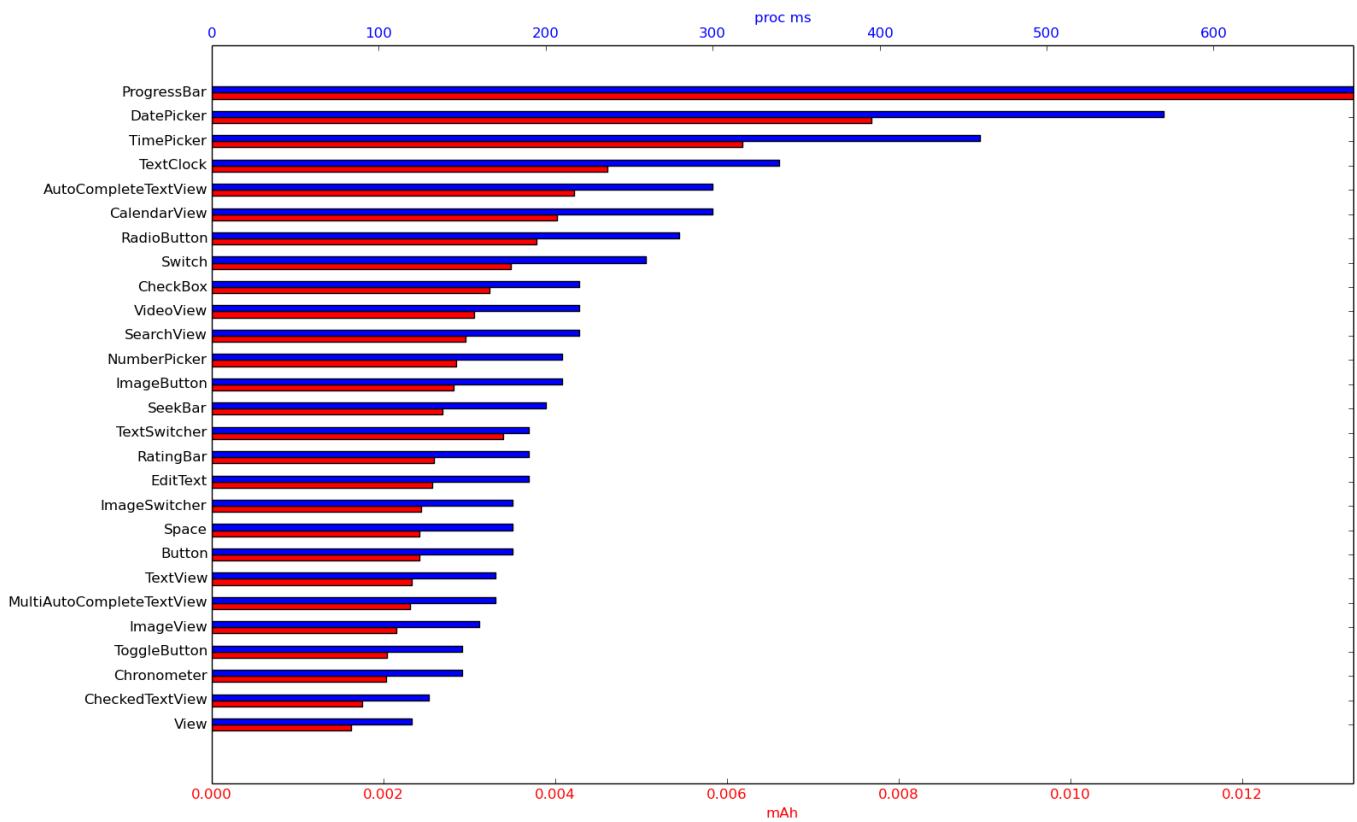


Рис. 16. Результаты измерений. Синие столбцы — процессорное время в мс.
Красные столбцы — количество затраченных мАч.

На графике неестественно выглядят показатели потребления TextSwitcher, так как они выбиваются из сортировки. Произошло это из-за выброса показателя потребления в одном из измерений, процессорное время в данном случае отображает более реальные данные.

На основании этих данных удалось составить рекомендацию по оптимизации для 23 виджетов из 27 представленных. Некоторые виджеты не имеют аналогов или уже являются наиболее оптимальными среди аналогичных.

10 Подключаемая библиотека

Теперь можно приступить к написанию библиотеки, которая будет являться конечным результатом работы.

10.1 Проектирование библиотеки

Перед написанием кода подключаемой библиотеки, необходимо её спроектировать. Проектирование поможет сделать работу библиотеки более стабильной и готовой к различным изменениям. К тому же, на хорошо спроектированные модули гораздо легче написать тесты при необходимости.

Задача проектирования заключается в продумывании архитектуры программного продукта, то есть описания того, из каких частей он будет состоять и как они будут между собой связаны. В моём случае библиотека выполняет следующие действия:

- прослушивание событий открытия нового экрана;
- прослушивание событий добавления новых элементов на текущий экран;
- сравнение заданного элемента интерфейса с его альтернативами на основании базы данных;
- вывод результата.

Изначально кажется, что первые две задачи очень похожи, но на деле это не совсем так, потому что за это отвечают разные механизмы и новый экран может быть открыт несколькими разными способами. Поэтому было решено разделить библиотеку на следующие модули:

- UIManager — точка входа в программу, которой необходим объект класса Application для установки слушателей открытия нового экрана. Здесь же определяются, какие реализации остальных компонентов будут использованы в работе.
- HierarchyAnalyzer — абстракция с методом analyzeDynamicHierarchy и полями типа Adviser и RecommendationOutputter. Этот класс будет слушать изменения в существующих иерархиях элементов, обращаясь к Adviser, чтобы сравнить элемент с имеющимися в базе данных и к RecommendationOutputter, чтобы вывести полученный результат.

- Adviser — абстракция с методом, который ищет оптимальный элемент, подобный заданному. Может возникнуть необходимость делать это асинхронно, так как поиск будет связан с работой с базой данных, поэтому метод `findAlternativeAsync` помимо имени класса элемента принимает callback, через который будет возвращён результат.
- RecommendationOutputter — абстракция с методом `output`, принимающим оригинальный элемент интерфейса и строку с описанием альтернативы.

Отношения между описанными частями представлены на рис. 17.

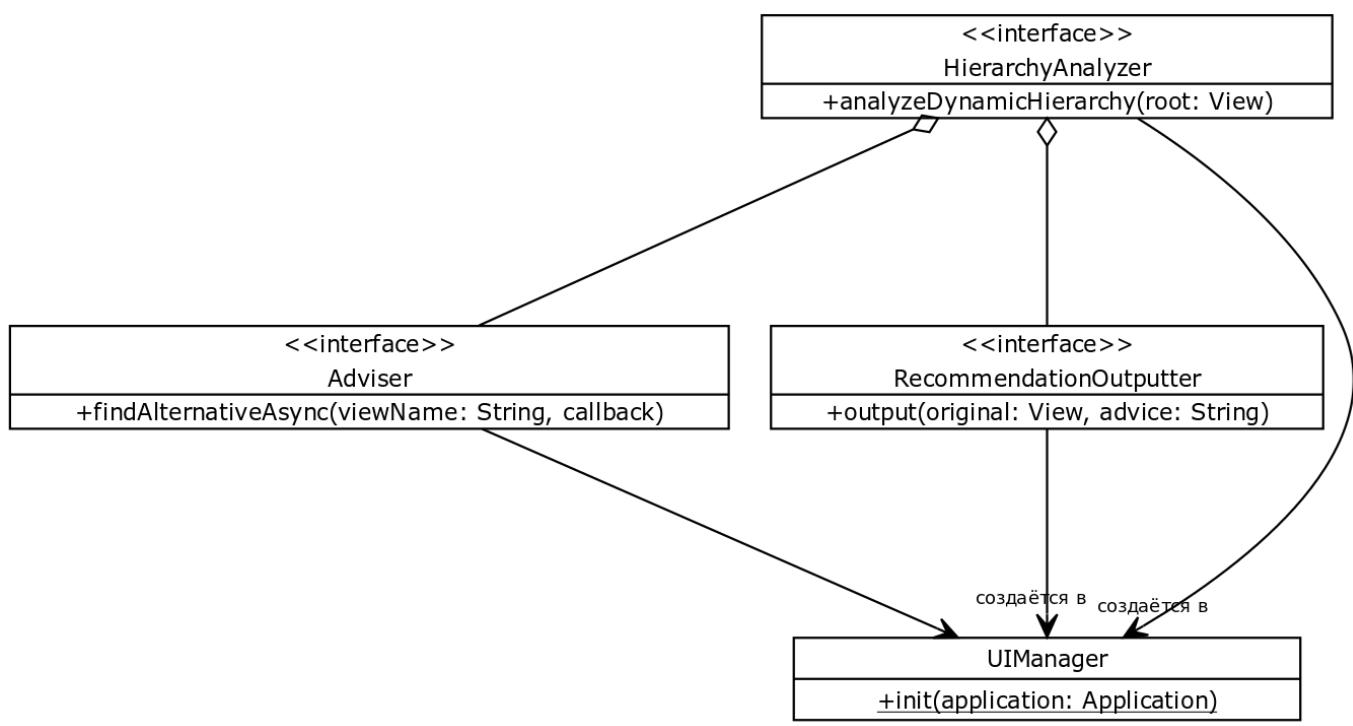


Рис. 17. Структурная диаграмма библиотеки

10.2 Язык программирования и инструментальные средства

Теперь нужно определиться, какими инструментами предстоит пользоваться при написании кода библиотеки. Самое основное — язык написания программы. Для разработки встраиваемой библиотеки необходимо использовать язык программирования Java и/или Kotlin. Дополнительно могут быть использованы модули, скомпилированные в .so файл, и написанные на любом языке, поддерживающим такую компиляцию.

Нет необходимости использовать подключение .so файлов в данном проекте, так как здесь не стоит задача обеспечения повышенной производительности и нет потребности в более тонком управлении памятью. Языки Kotlin и Java не имеют разницы в производительности, так как в итоге компилируются в одинаковый байт-код. Различия имеются только в удобстве написания, здесь в большинстве случаев более оптимален Kotlin, позволяющий писать более короткий и более читаемый для человека код. Так как эти языки совместимы, при необходимости можно будет написать фрагмент программы на языке Java, но основным языком разработки был выбран Kotlin.

Так как языки Java и Kotlin совместимы, для написания собственной библиотеки можно использовать другие библиотеки, написанные как на языке Java, так и на языке Kotlin. Сторонние библиотеки могут упростить прослушивание событий, работу с базой данных, реализацию алгоритмов сравнения элементов друг с другом и так далее. Но чтобы не увеличивать размер библиотеки и не добавлять в неё функциональность из сторонней библиотеки, которая не будет использована, было решено пользоваться только средствами Android SDK, а также языков Java и Kotlin.

В библиотеке предстоит отслеживать смену экранов и динамическое изменение иерархий элементов, в Android SDK для этого имеются такие средства как Application.ActivityLifecycleCallbacks, FragmentManager.FragmentLifecycleCallbacks и ViewGroup.OnHierarchyChangeListener. ActivityLifecycleCallbacks используются для отслеживания изменений состояния жизненного цикла Activity. Его удобно использовать, чтобы определять, что создалась новая сущность Activity (для этой сущности будет вызван метод жизненного цикла onCreate). FragmentLifecycleCallbacks может использоваться для отслеживания жизненного цикла Fragment (компонент, который может представлять собой отдельный экран или его часть). OnHierarchyChangeListener в свою очередь позволяет отслеживать динамические изменения иерархии элементов графического интерфейса.

Так как ViewGroup.OnHierarchyChangeListener полностью покрывает все текущие сценарии использования FragmentManager.FragmentLifecycleCallbacks, было решено использовать Application.ActivityLifecycleCallbacks для определения появления новых Activity вместе с ViewGroup.OnHierarchyChangeListener, чтобы находить новые сущности Fragment и вручную добавляемые программистом представления.

10.3 База данных

Базу данных решено было спроектировать следующим образом. Существует одна таблица с тремя столбцами. В первом столбце содержится `id` строки, это поле обязательно. Второй столбец содержит название виджета, для которого в третьем столбце находится название самого близкого по потреблению аналога. Идея состоит в том, чтобы получать самый близкий аналог, потребляющий меньше текущего и выполнять новый поиск для полученного аналога. Таким образом, поиск будет производиться до тех пор, пока не будет получен виджет, не имеющий более энергоэффективных аналогов. Результатом станет полный список виджетов, которыми можно заменить текущий виджет, чтобы повысить энергоэффективность приложения.

Иногда нужно дописать какое-либо дополнение к замене, в таких случаях первым словом будет название виджета, а после него дополнение, которое будет добавляться к каждому последующему виджету.

База данных содержит рекомендации по замене следующих виджетов:

- `ProgressBar` — `Switch`
- `DatePicker` — `CalendarView`
- `TimePicker` — `AutoCompleteTextView`
- `AutoCompleteTextView` — `EditText`
- `CalendarView` — `EditText`
- `RadioButton` — `Switch` with custom logic
- `Switch` — `CheckBox`
- `CheckBox` — `TextSwitcher`
- `SearchView` — `EditText`
- `NumberPicker` — `SeekBar`
- `ImageButton` — `ImageSwitcher`
- `SeekBar` — `EditText`

- TextSwitcher — ImageSwitcher
- RatingBar — EditText
- EditText — MultiAutoCompleteTextView
- ImageSwitcher — ToggleButton
- Space — View
- Button — TextView
- TextView — ImageView
- MultiAutoCompleteTextView — ToggleButton
- ImageView — CheckedTextView
- ToggleButton — CheckedTextView
- CheckedTextView — View with background

Созданная база данных помещена в директорию `assets` модуля `hierarchy-checker`, чтобы иметь доступ к бинарному файлу базы из библиотеки.

10.4 Разработка

Перед началом написания кода библиотеки встаёт вопрос организации кода.

10.4.1 Структура проекта

Сам проект будет удобно разделить на несколько модулей, чтобы хранить весь код в одном проекте. Модуль `estimation` содержит весь код, необходимый для проведения измерений. Модуль `hierarchy-checker` содержит код встраиваемой библиотеки. Модуль `example` подключает модуль `hierarchy-checker` и демонстрирует работу библиотеки.

Модуль `estimation` в корневой директории содержит все используемые скрипты, а также результаты измерений. В директории `src/androidTest` содержится класс `MyTestRunner`, а также код автотестов. В директории `src/main` находятся наследники `ViewWrapper`, а также класс `MainActivity`.

10.4.2 Написание кода библиотеки

При написании кода было решено начать с описания спроектированной архитектуры в синтаксисе языка программирования Kotlin. UIManager — singleton-объект, содержащий методы init и getActivityRoot. Первый метод — точка входа в приложение, он принимает объект приложения и ничего не возвращает. getActivityRoot принимает на вход объект Activity и возвращает корневое представление иерархии, привязанное к этому Activity.

Adviser — интерфейс с методом findAlternativeAsync, который принимает имя класса представления, которое необходимо проанализировать, а также callback, через который будет возвращён результат. RecommendationOutputer тоже интерфейс с методом output, принимающим объект View, для которого сформирована рекомендация, и сама рекомендация в виде строки.

HierarchyAnalyzer — абстрактный класс, у которого имеется конструктор, принимающий реализации Adviser и RecommendationOutputer и сохраняющий их в поля на уровне абстрактного класса. Также он содержит абстрактный метод analyzeDynamicHierarchy, который принимает корень иерархии элементов, которую необходимо проанализировать.

Реализация HierarchyAnalyzer содержит объект OnHierarchyChangeListener, который она передаёт всем объектам ViewGroup, встречающимся в иерархии. Для прохода иерархии используется итерация по всем дочерним элементам корневого элемента и рекурсивного вызова analyzeDynamicHierarchy в случае, если дочерний элемент также может содержать дочерние элементы. Реализация OnHierarchyChangeListener вызывает метод analyzeDynamicHierarchy для каждого динамически добавленного элемента.

В качестве реализации интерфейса RecommendationOutputer используется LogOutputer, выводящий информацию в логи устройства. При вызове метода output реализация проходит по всем родительским элементам, чтобы составить полный адрес текущего элемента в иерархии. Для смены порядка родительских элементов (идём снизу вверх по иерархии, выводим её сверху вниз) используется стек. После этого формируется сообщение, содержащее класс объекта, его расположение в иерархии и совет по оптимизации. Сообщение выводится в логи устройства.

Класс DatabaseAdviser реализует интерфейс Adviser и содержит в поле класса объект AdviceHelper, который абстрагирует взаимодействие с базой данных, а также ThreadPoolExecutor, который сохраняет несколько потоков исполнения, которые

могут быть многоразово использованы или завершены, если дополнительных задач не поступает в течение 30 секунд. Предполагается, что его использование сократит издержки на создание новых потоков за счёт переиспользования уже существующих.

При вызове метода `findAlternativeAsync` на потоке, который предоставляет объект `ThreadPoolExecutor`, вызывается метод `getAdvice` у `AdviceHelper`, который возвращает более эффективный аналог. Но чтобы сформировать наиболее полный список аналогов, метод `getAdvice` вызывается ещё раз для результата предыдущего вызова. Так происходит до тех пор, пока программа не достигнет самого оптимального виджета. Все виджеты будут описаны в результате, который уже на главном потоке передаётся в переданный методу `callback`. В иерархии реального приложения некоторые названия виджетов могут отличаться от того, что есть в базе данных. Например, вместо `TextView` можно встретить `AppCompatTextView`. Такая замена совершается операционной системой и предусматривается алгоритмами поиска в базе данных.

Класс `AdviceHelper` наследует `SQLiteOpenHelper` и инкапсулирует всю работу с базой данных. Одна из главных задач, которую он решает, это использование заранее готовой базы данных. Изначально база данных находится в специальном месте для бинарных файлов, но использовать её из этой директории нельзя, так как `SQLiteOpenHelper` работает только с базами, созданными самим приложением, и находящимися в отдельной директории. Чтобы решить эту задачу, при инстанцировании `AdviceHelper` проверяет, существует ли база данных, созданная приложением. Если такой нет, значит запуск библиотеки с этим приложением происходит впервые. В этом случае файл базы копируется из изначальной директории в директорию с базами данных приложения. Если база уже существует, открывается соединение с ней для проверки её версии. Если версия совпадает с ожидаемой, база закрывается, если нет, база, возможно, устарела. В случае устаревшей базы данных, она удаляется и заново копируется из директории с оригинальным файлом, который мог измениться.

В методе `init` класса `UIManager` создаются конкретные реализации `Adviser` и `RecommendationOutputter`, они передаются в конструктор `ConcreteHierarchyAnalyzer`. Затем на объекте приложения вызывается `registerActivityLifecycleCallbacks`, а в реализации `Application.ActivityLifecycleCallbacks` при приобретении какой-либо `Activity`

с состояния Started, с помощью метода getActivityRoot находится корень его иерархии и передаётся в метод analyzeDynamicHierarchy созданного ранее HierarchyAnalyzer.

Итоговая библиотека имеет структуру, представленную на рис 18.

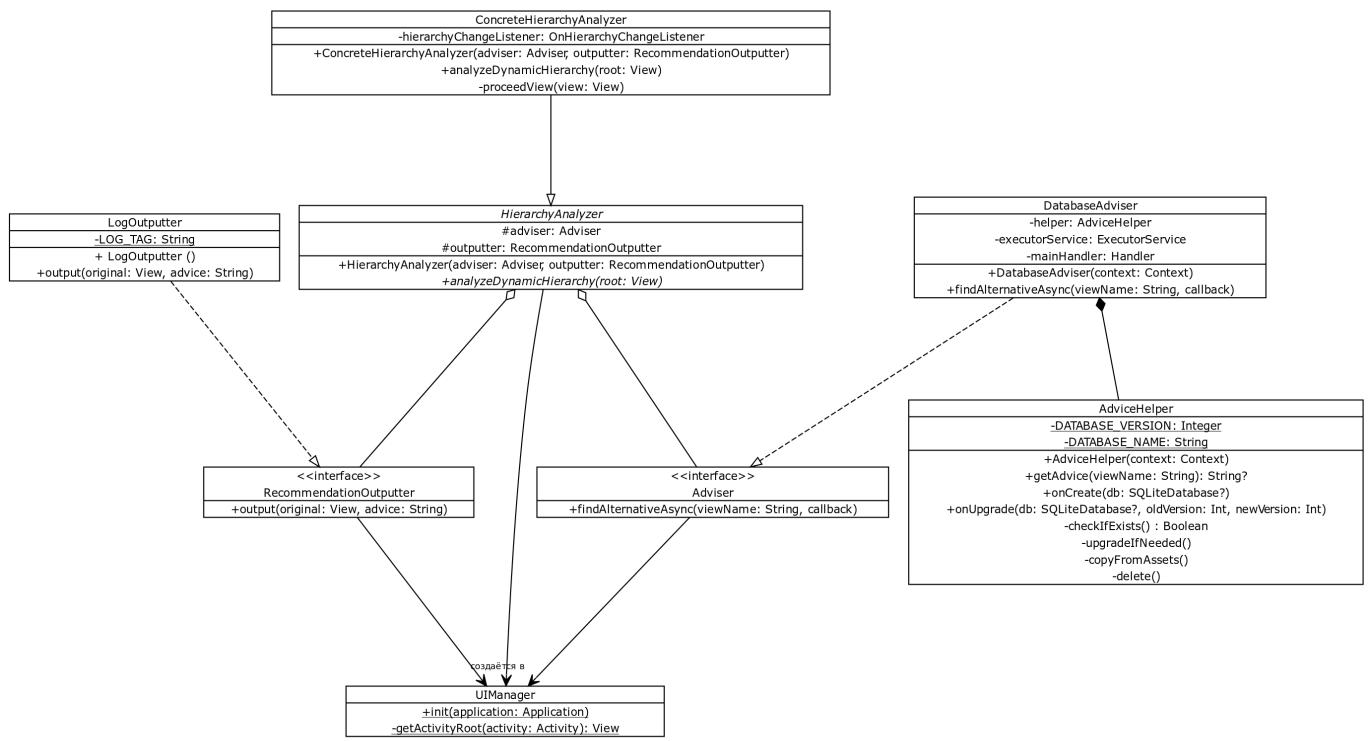


Рис. 18. Диаграмма классов библиотеки

11 Итоги работы

Весь исходный код проекта, а также необработанные результаты всех проведённых измерений были опубликованы на GitHub [28].

Для демонстрации работы библиотеки в проект был добавлен модуль **example**. Модуль содержит приложение, способное динамически добавлять на экран объект класса **android.app.Fragment** с наполнением, объект класса **androidx.fragment.app.Fragment** с наполнением, объекты класса **TextView**, а также открывать новое Activity (рис. 19). К проекту подключена библиотека из модуля **hierarchy-checker** и в методе `onCreate` наследника класса Application у объекта `UIManager` был вызван метод `init`, в который был передан контекст приложения.

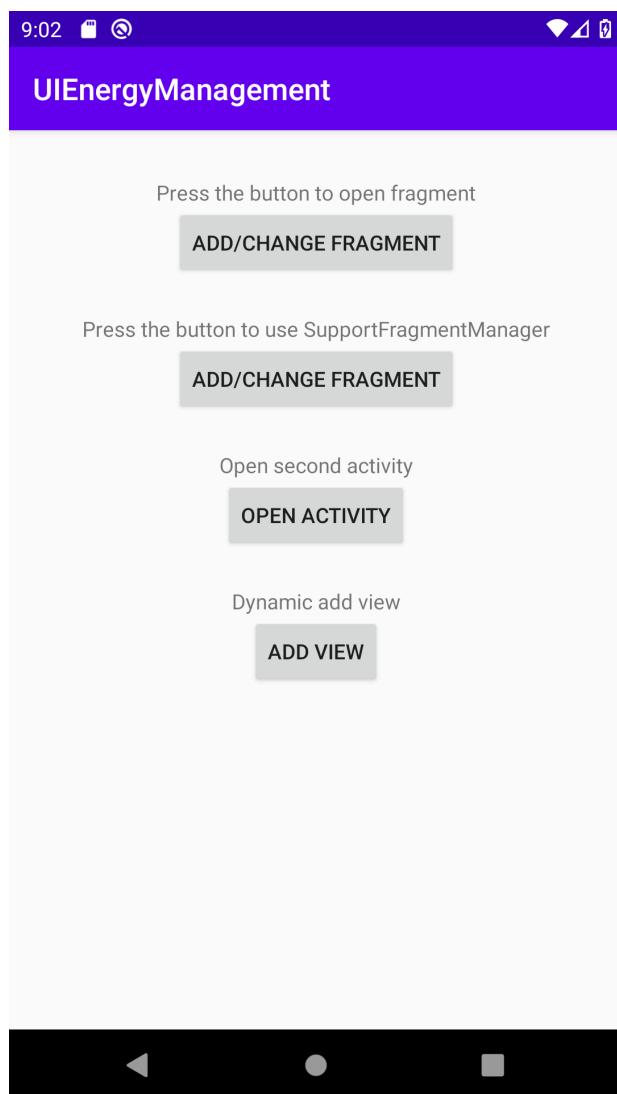


Рис. 19. Интерфейс приложения из модуля example

Так как главный экран приложения уже содержит кнопки и текстовые поля, сразу после открытия приложения в логах устройства появляются советы по

оптимизации открывшегося экрана (рис. 20).

```
out : AppCompatTextView could be replaced by (less effective to more effective): ImageView, CheckedTextView, View with background
extView could be replaced by (less effective to more effective): ImageView, CheckedTextView, View with background
: AppCompatButton could be replaced by (less effective to more effective): TextView, ImageView, CheckedTextView, View with background
ut : AppCompatTextView could be replaced by (less effective to more effective): ImageView, CheckedTextView, View with background
: AppCompatButton could be replaced by (less effective to more effective): TextView, ImageView, CheckedTextView, View with background
: AppCompatButton could be replaced by (less effective to more effective): TextView, ImageView, CheckedTextView, View with background
ut : AppCompatTextView could be replaced by (less effective to more effective): ImageView, CheckedTextView, View with background
```

Рис. 20. Логи устройства после открытия приложения из модуля example

При нажатии кнопок на экран добавляются новые виджеты или открывается новый экран. При этом библиотека находит новые виджеты в иерархии и сразу после их отображения выводит в логи устройства советы по оптимизации, если их удалось найти в базе данных.

ЗАКЛЮЧЕНИЕ

Для разработки системы контроля и управления энергопотреблением элементов графического интерфейса на устройствах под управлением операционной системы Android были проведены измерения 27 виджетов в пакете android.widget в Android 10 SDK Platform. Результаты измерений показали соотношения энергопотребления между всеми измеренными элементами.

На основании результатов измерений была составлена база данных, которая используется встраиваемой библиотекой для формирования советов по уменьшению энергопотребления приложения. Это позволит разработчикам приложений выявлять неэффективные с точки зрения энергопотребления виджеты приложения получать советы по их оптимизации.

Существенными ограничениями данной работы являются следующие факторы:

- Отсутствие доступа к стороннему оборудованию для измерения энергопотребления;
- Отсутствие доступа к устройству без сторонних приложений и без служб Google Play Services;
- Наличие лишь одного устройства для тестирования.

Изначально для измерения энергопотребления устройства планировалось использовать специализированное оборудование Monsoon Power Monitor. Использование данного оборудования позволило бы более точно измерить энергопотребление устройства, причём не за всё время тестирования виджета, а за более малые промежутки. Инструмент Battery Historian также имеет возможность обработки результатов работы Monsoon Power Monitor. В дальнейших исследованиях можно повторить измерения с использованием более точного стороннего оборудования.

Отсутствие сторонних приложений на тестируемом устройстве могло бы снизить искажения проводимых измерений, так как приложения могут совершать фоновую работу. Часть затраченных ресурсов операционная система разделяет на все запущенные в данный момент приложения и результаты измерений могут быть искажены. Службы Google Play Services — одна из главных составляющих фонового потребления ресурсов устройства, поэтому тестирование оптимальнее проводить на

устройстве без данных служб. Их отличие от приложений состоит в том, что службы предустановлены в операционную систему и не могут быть удалены или отключены обычным пользователем.

Некоторые искажения могут быть связаны с моделью устройства, используемого для тестирования, или с версией операционной системы, установленной на нём. Для получения более объективных показателей требуется проведение тестирования на нескольких устройствах и обобщение результатов.

В дальнейших исследованиях может быть увеличено количество проводимых измерений, виджеты во время тестирования могут перерисовываться вызовом метода `invalidate`, а также может быть сымитировано пользовательское взаимодействие с устройством.

ГЛОССАРИЙ

База данных — организованная структура, которая используется для хранения, обработки и изменения взаимосвязанной информации.

Библиотека — набор ресурсов, используемых для разработки программного обеспечения.

Виджет — элемент графического интерфейса, используемый на экране приложения.

Фреймворк автоматического тестирования — программное обеспечение, позволяющее создавать тесты для автоматизированного тестирования приложения. Частные случаи: appium, espresso, kakaо, kaspresso.

Activity — компонент приложения, имеющий графический интерфейс, и позволяющий пользователю с ним взаимодействовать.

Android — мобильная операционная система с открытым исходным кодом.

Android Debug Bridge — инструмент командной строки, позволяющий взаимодействовать с устройством.

Android Profiler — инструмент, предоставляющий данные о потребляемых приложением ресурсах устройства в реальном времени.

Battery Historian — инструмент анализа и визуализации расхода батареи устройства на основании отчётов операционной системы.

BroadcastReceiver — компонент приложения, позволяющий принимать и обрабатывать системные события.

Uid — пользовательский идентификатор в Unix-подобных операционных системах, он уникален для каждого приложения и по нему можно отслеживать потребление.

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ

Автотест — тест для автоматизированного тестирования приложения.

ОС — операционная система.

adb — Android Debug Bridge.

API — application program interface.

mAh — миллиампер-час.

zsh — Z shell.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Experience report: Understanding cross-platform app issues from user reviews / Y. Man [и др.] // 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE). — IEEE. 2016. — С. 138—149.
2. Wasserman A. I. Software engineering issues for mobile application development // Proceedings of the FSE/SDP workshop on Future of software engineering research. — 2010. — С. 397—400.
3. Ickin S., Petersen K., Gonzalez-Huerta J. Why do users install and delete Apps? A survey study // International Conference of Software Business. — Springer. 2017. — С. 186—191.
4. Diversity in smartphone usage / H. Falaki [и др.] // Proceedings of the 8th international conference on Mobile systems, applications, and services. — 2010. — С. 179—194.
5. Pentikousis K. In search of energy-efficient mobile networking // IEEE Communications Magazine. — 2010. — Т. 48, № 1. — С. 95—103.
6. Василенко Д., Бирюков Е. Методы снижения потребления энергии современными портативными устройствами // Компоненты и Технологии. — 2005. — № 50.
7. Tuysuz M. F., Ucan M., Trestian R. A real-time power monitoring and energy-efficient network/interface selection tool for android smartphones // Journal of Network and Computer Applications. — 2019. — Т. 127. — С. 107—121.
8. Li D., Halfond W. G. J. An investigation into energy-saving programming practices for android smartphone app development // Proceedings of the 3rd International Workshop on Green and Sustainable Software. — 2014. — С. 46—53.
9. Android power management and analyses of power consumption in an Android smartphone / G. Bai [и др.] // 2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing. — IEEE. 2013. — С. 2347—2353.
10. Detecting display energy hotspots in Android apps / M. Wan [и др.] // 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST). — IEEE. 2015. — С. 1—10.

11. Improving Energy Efficiency of Android Devices by Preventing Redundant Frame Generation / G. Lee [и др.] // IEEE Transactions on Mobile Computing. — 2018. — Т. 18, № 4. — С. 871—884.
12. Intelligent frame refresh for energy-aware display subsystems in mobile devices / Y. Huang [и др.] // Proceedings of the 2014 international symposium on Low power electronics and design. — 2014. — С. 369—374.
13. AppsScope: Application energy metering framework for android smartphone using kernel activity monitoring / C. Yoon [и др.] // Presented as part of the 2012 {USENIX} Annual Technical Conference ({USENIX} {ATC} 12). — 2012. — С. 387—400.
14. Seo C., Malek S., Medvidovic N. An energy consumption framework for distributed java-based systems // Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering. — 2007. — С. 421—424.
15. Mobile application and device power usage measurements / R. Murmuria [и др.] // 2012 IEEE Sixth International Conference on Software Security and Reliability. — IEEE. 2012. — С. 147—156.
16. Oliveira W., Oliveira R., Castor F. A study on the energy consumption of android app development approaches // 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR). — IEEE. 2017. — С. 42—52.
17. Экономия энергии посредством компрессии во встроенной среде Java / Ч. Гуанджиу [и др.] // Компоненты и Технологии. — 2004. — № 39.
18. Maurycio C. Переоценка подхода к снижению потребления энергии // Компоненты и Технологии. — 2008. — № 82.
19. Пустовалов Е. В., Тюриков А. М. Анализ режимов энергосбережения мобильного пользовательского устройства // Известия высших учебных заведений. Приборостроение. — 2013. — Т. 56, № 8.
20. Кафтанников И. Л., Руднев В. А. Оптимизация энергопотребления микроконтроллерных систем // Вестник Южно-Уральского государственного университета. Серия: Компьютерные технологии, управление, радиоэлектроника. — 2013. — Т. 13, № 2.

21. An empirical study of the energy consumption of android applications / D. Li [и др.] // 2014 IEEE International Conference on Software Maintenance and Evolution. — IEEE. 2014. — С. 121—130.
22. Адаптивное управление приложениями мобильных телефонов для увеличения продолжительности их автономной работы / Л. Л. Утин [и др.] // Доклады Белорусского государственного университета информатики и радиоэлектроники. — 2018. — 2 (112).
23. Multi-objective optimization of energy consumption of GUIs in Android apps / M. Linares-Vásquez [и др.] // ACM Transactions on Software Engineering and Methodology (TOSEM). — 2018. — Т. 27, № 3. — С. 1—47.
24. *Kim D., Jung N., Cha H.* Content-centric display energy management for mobile devices // 2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC). — IEEE. 2014. — С. 1—6.
25. Bugreport file format. — URL: <https://android.googlesource.com/platform/frameworks/native/+master/cmds/dumpstate/bugreport-format.md> (дата обр. 10.05.2020).
26. dumpsys. — URL: <https://developer.android.com/studio/command-line/dumpsys> (дата обр. 10.05.2020).
27. Manifest.permission.DUMP. — URL: <https://developer.android.com/reference/android/Manifest.permission#DUMP> (дата обр. 10.05.2020).
28. *Yundin V.* UIEnergyManagement. — URL: <https://github.com/Yundin/UIEnergyManagement> (дата обр. 15.05.2020).

Приложение

Список сущностей пакета android.widget в Android 10 SDK Platform

- AbsListView
- AbsoluteLayout
- AbsSeekBar
- AbsSpinner
- ActionMenuView
- Adapter
- AdapterView
- AdapterViewAnimator
- AdapterViewFlipper
- Advanceable
- AlphabetIndexer
- AnalogClock
- ArrayAdapter
- AutoCompleteTextView
- BaseAdapter
- BaseExpandableListAdapter
- Button
- CalendarView
- Checkable
- CheckBox
- CheckedTextView

- Chronometer
- CompoundButton
- CursorAdapter
- CursorTreeAdapter
- DatePicker
- DialerFilter
- DigitalClock
- EdgeEffect
- EditText
- ExpandableListAdapter
- ExpandableListView
- Filter
- Filterable
- FilterQueryProvider
- FrameLayout
- Gallery
- GridLayout
- GridView
- HeaderViewListAdapter
- HeterogeneousExpandableList
- HorizontalScrollView
- ImageButton

- ImageSwitcher
- ImageView
- LinearLayout
- ListAdapter
- ListPopupWindow
- ListView
- Magnifier
- MediaController
- MultiAutoCompleteTextView
- NumberPicker
- OverScroller
- PopupMenu
- PopupWindow
- ProgressBar
- QuickContactBadge
- RadioButton
- RadioGroup
- RatingBar
- RelativeLayout
- RemoteViews
- RemoteViewsService
- ResourceCursorAdapter

- ResourceCursorTreeAdapter
- Scroller
- ScrollView
- SearchView
- SectionIndexer
- SeekBar
- ShareActionProvider
- SimpleAdapter
- SimpleCursorAdapter
- SimpleCursorTreeAdapter
- SimpleExpandableListAdapter
- SlidingDrawer
- Space
- Spinner
- SpinnerAdapter
- StackView
- Switch
- TabHost
- TableLayout
- TableRow
- TabWidget
- TextClock

- `TextSwitcher`
- `TextView`
- `ThemedSpinnerAdapter`
- `TimePicker`
- `Toast`
- `ToggleButton`
- `Toolbar`
- `TwoLineListItem`
- `VideoView`
- `ViewAnimator`
- `ViewFlipper`
- `ViewSwitcher`
- `WrapperListAdapter`
- `ZoomButton`
- `ZoomButtonsController`
- `ZoomControls`