Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Алтайский государственный технический университет им. И.И. Ползунова»
Факультет информационных технологий
Кафедра "Прикладная математика"

Курсовой проект защищен с оценкой _____

Преподаватель _____ С.М. Старолетов
Подпись
«_____»_____2017 г.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ

Проектирование системы проведения анализа вредоносного программного обеспечения

по дисциплине «Архитектурное проектирование и паттерны программирования»

КП 09.03.04.07.000 ПЗ

Студент группы ПИ-42 _____ Д.С. Гутяр
(подпись) и.о.,фамилия

Преподаватель доцент, к. ф-м н. _____ С.М. Старолетов
(должность, ученая степень) (подпись) и.о.,фамилия

Барнаул 2017

Задание

Учебная дисциплина: Архитектурное проектирование и паттерны программирования

ФИО студента: Гутяр Дмитрий Сергеевич

Группа: ПИ-42

Тема курсового проекта: Проектирование системы проведения анализа вредоносного программного обеспечения

Этапы разработки курсового проекта и сроки их выполнения:

1. Изучение необходимой учебной и научно-технической литературы (01.10.2017 – 30.10.2017);
2. Разработка приложения (30.10.2017 – 20.11.2017);
3. Оформление отчета о проделанной работе (20.11.2017 – 20.12.2017);
4. Сдача работы руководителю и защита работы (20.12.2017 – 28.12.2017).

Дата выдачи задания: 01.10.2017

Срок защиты: 28.12.2017

Руководитель: доцент С.М. Старолетов

					КП 09.03.04.07.000 ПЗ			
Изм.	Лист	№ докум.	Подп.	Дата				
Разраб.		Гутяр Д.С.			Проектирование системы проведения анализа вредоносного программного обеспечения	Лит.	Лист	Листов
Пров.		Старолетов С.М.				У	2	29
						АлтГТУ, ФИТ *гр. ПИ-42*		
Н.контр.		Старолетов С.М.						
Утв.		Кантор С.А						

Содержание

Введение

В настоящее время электронные вычислительные устройства подвергаются миллионам вирусных атак. При этом должно пройти некоторое время для того, чтобы эти вирусные угрозы были обнаружены и база антивирусных средств обновилась. За это время злоумышленники могут похитить или зашифровать ценную или конфиденциальную информацию, которую можно использовать для давления на жертву атаки (например, в целях вымогательства), либо для нанесения ущерба фирме, владеющей этими устройствами. С повсеместным распространением мобильных устройств также преумножилось и количество вирусов, написанных специально под мобильные платформы. Для наиболее быстрого определения опасности используемой программы требуется локальное средство, позволяющее собрать статистику по приложению и проанализировать.

1 Анализ проблемы и существующих методов решения

1.1 Обзор предметной области

Существует множество вредоносных программ, причем их с каждым днем становится больше. Они различаются по способам воздействия и могут похищать, зашифровывать данные пользователя, либо понижать производительность системы (в том числе коммерческой) для нанесения ущерба, либо вымогательства. Зачастую, чтобы затруднить анализ вредоносного ПО, сначала на устройство жертвы загружается небольшая программа, которая скачивает основной вирус с серверов злоумышленников. Он, в свою очередь, проверяет файловую систему на наличие файлов, потенциально полезных для мошенничества. Например, ищет банковские клиенты, после чего выполняет чтение данных из их файлов и отправляет на сервер. Таким образом злоумышленник может получить данные для удаленного доступа к деньгам жертвы.

При этом первая программа может устанавливаться под видом каких-либо системных обновлений и без предупреждений получать системные права и права на удаление всех данных на мобильном устройстве [1]. Также есть загрузчики, которые собирают информацию об уязвимостях системы, чистят логи за собой и самоудаляются. Таким образом, вся информация о безопасности системы пересылается преступникам, а данные о способе проникновения уничтожаются и система остается уязвимой для повторного заражения [2].

По мере распространения мобильных устройств они заняли большую нишу в жизни каждого человека. Теперь на них хранится множество личной информации, начиная от фотографий, переписки, заканчивая паролями от банковских карт и данными для авторизации в банковских службах, также, используя базовые возможности телефона, можно переводить деньги со счета, собирать еще больше личной информации, либо использовать вычислительные мощности телефона в качестве ячейки ботсети. Помимо прочего, на многих не установлен антивирус и система в принципе уязвима к воздействиям вредоносных приложений. Это делает их

очень привлекательными для злоумышленников. Примерами функций, которые обладают вирусные приложения, могут быть:

- нежелательная (навязчивая) реклама
- загрузка нежелательных приложений
- подписка пользователей на платные услуги путем отправки смс с подтверждением оплаты
- использование телефона в качестве звена DDoS-атаки на веб-ресурсы
- майнинг криптовалют

Последние две функции вредят также и физическому состоянию устройства, поскольку загружают его на максимальном уровне и продолжают использовать длительное время (особенно майнинг) [3].

Все эти функции вирусов неприятны пользователю, но избавиться от мобильного вируса после заражения бывает практически нельзя. Притом понять, какое приложение является вирусом, а какое нет тоже довольно сложно, поскольку вирусы максимально маскируются под необходимые системные, либо очень полезные приложения. Чтобы однозначно определить опасность ПО потребуется провести его комплексный анализ, включающий проверку прав приложения и построение графа состояний. Граф состояний (пример на рисунке 1.1) позволяет исследователю быстро оценить, как работает приложение, какие функции оно использует и каким образом.

Граф состояний также используется в процессе разработки, чтобы наглядно уяснить, какие действия из каких условий должны вытекать при работе программы. Граф состояний вируса показан на рисунке 1.

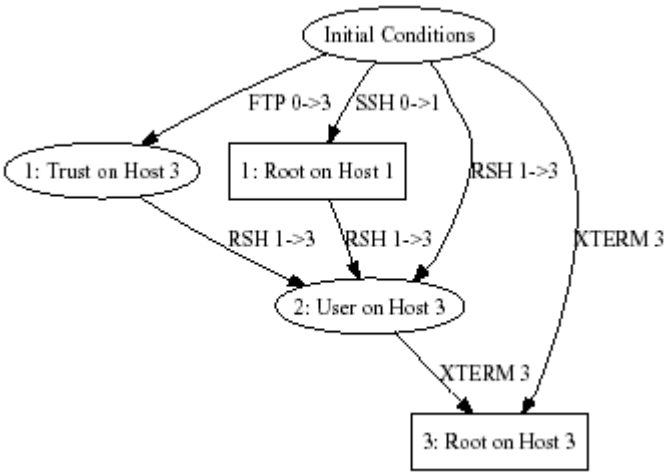

Рисунок 1.1 – Граф состояний вируса

Сложность в построении такого графа автоматически заключается в том, что при реверсном инжиниринге изначально неизвестна роль того или иного метода в программе. Поэтому полноценный граф, отражающий функционал ПО можно отобразить только после его полного изучения вручную. При большом количестве вирусов это становится невозможным. С другой стороны, можно выделить методы программы в качестве ее состояний и отобразить основные переходы между ними, произведя синтаксический анализ кода.

1.2 Анализ существующих решений

Для проведения анализа ПО на предмет вредоносности зачастую используется несколько продуктов, обладающих разными возможностями. Такой список продуктов включает:

- Декомпилятор
- Архиватор
- Эмулятор
- Прокси-сервер
- Система логирования

Декомпилятором расшифровывается исходный apk файл и появляется возможность просмотреть код приложения и структуру классов в нем. В качестве декомпилятора может использоваться dex2jar, это бесплатный набор библиотек для расшифровки. Но файлы могут быть специально зашифрованы, поэтому лучше стоит иметь несколько GUI, предлагающих разные приемы расшифровки. Примерами GUI могут быть:

- Java Decompiler, который может использоваться как отдельное приложение, либо в качестве плагина на среду разработки
- Java DeCompiler – Luyten
- DJ Java Decompiler, мощное средство, о котором можно сказать отдельно

DJ Java Decompiler - это декомпилятор и дизассемблер для Windows XP, Windows 2003, Windows Vista, Windows 7, Windows 8, 8.1 и 10 для Java, который восстанавливает исходный код из скомпилированных двоичных файлов CLASS (например, Java-апплетов). DJ Java Decompiler способен декомпилировать сложные Java-апплеты и двоичные файлы, создавая точный исходный код. Он позволяет

быстро получить всю необходимую информацию о файлах классов. Он является автономным приложением Windows; при этом не требует наличия Java. Также обладает возможностями редактора Java. Для его использования не требуется устанавливать виртуальную машину Java или любой другой Java SDK. [4]

Архиватор используется для просмотра списка файлов внутри архива apk или его разархивирования. Примером архиватора может служить **WinRAR.**

Так как вирусы могут приводить к необратимым процессам в системе, использовать физическое устройство для их тестирование нерационально, так как оно может быть повреждено даже физически вследствие продолжительного воздействия высоких нагрузок на процессор, например, может вздуться батарея[3]. Поэтому для таких целей используются эмуляторы, на которых устанавливается вирусное приложение и исследуется его работа. В качестве примера можно указать **Genymotion Android Emulator.** Там же можно в настройках сети указать Proxy-сервер, который будет отслеживать web-трафик. Зачастую Genymotion используется в связке с **Android Studio** для отслеживания логов.

Также для проверки потенциально вредоносного ПО используется средство **VirusTotal.** VirusTotal - бесплатная служба, осуществляющая анализ подозрительных файлов и ссылок (URL) на предмет выявления вирусов, червей, троянов и всевозможных вредоносных программ. В его базе содержится множество вирусных сигнатур, которые постоянно пополняются.

Существуют средства, использующие модифицированное ядро android для наблюдения за вирусами и сбором статистической информации (например, **droidbox**). Модификация заключается в сборе логов приложений. После прерывания его работы информация обрабатывается и на выходе получаются графики, показывающие время работы в определенном режиме, таком, как:

- Входящие / исходящие сетевые данные
- Операции чтения и записи файлов

- Запущенные службы и загруженные классы через DexClassLoader
- Утечка информации через сеть, файлы и SMS
- Ограниченные разрешения
- Криптографические операции, выполняемые с использованием API Android
- Список широковещательных приемников
- Отправленные SMS и телефонные звонки

Генерируются два графика визуализации поведения пакета. Один из них показывает временный порядок операций (указан на рисунке 1.2), а другой – treemap (на рисунке 1.3), который может использоваться для проверки сходства между проанализированными пакетами.

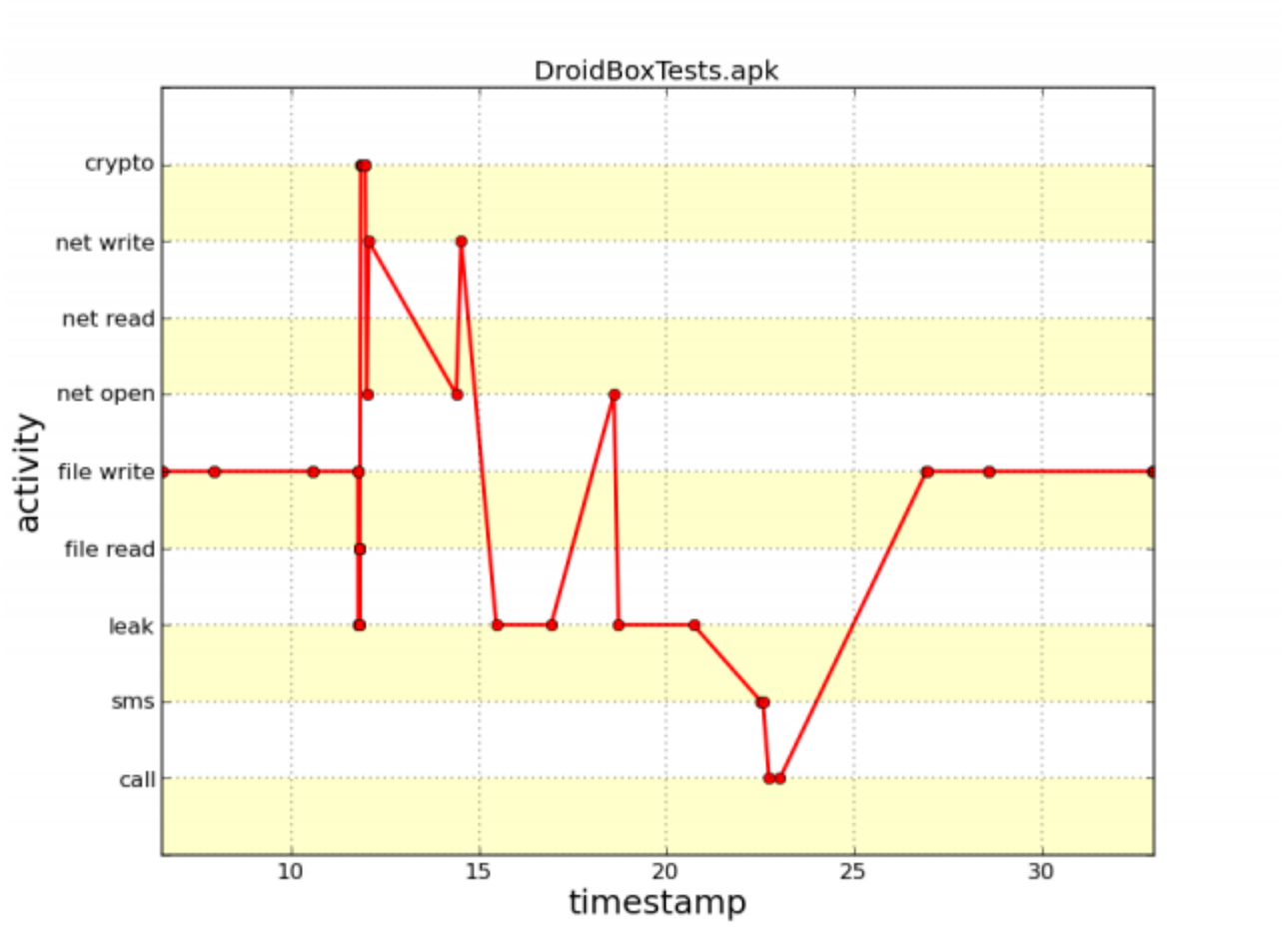

Рисунок 1.2 – График активности приложения

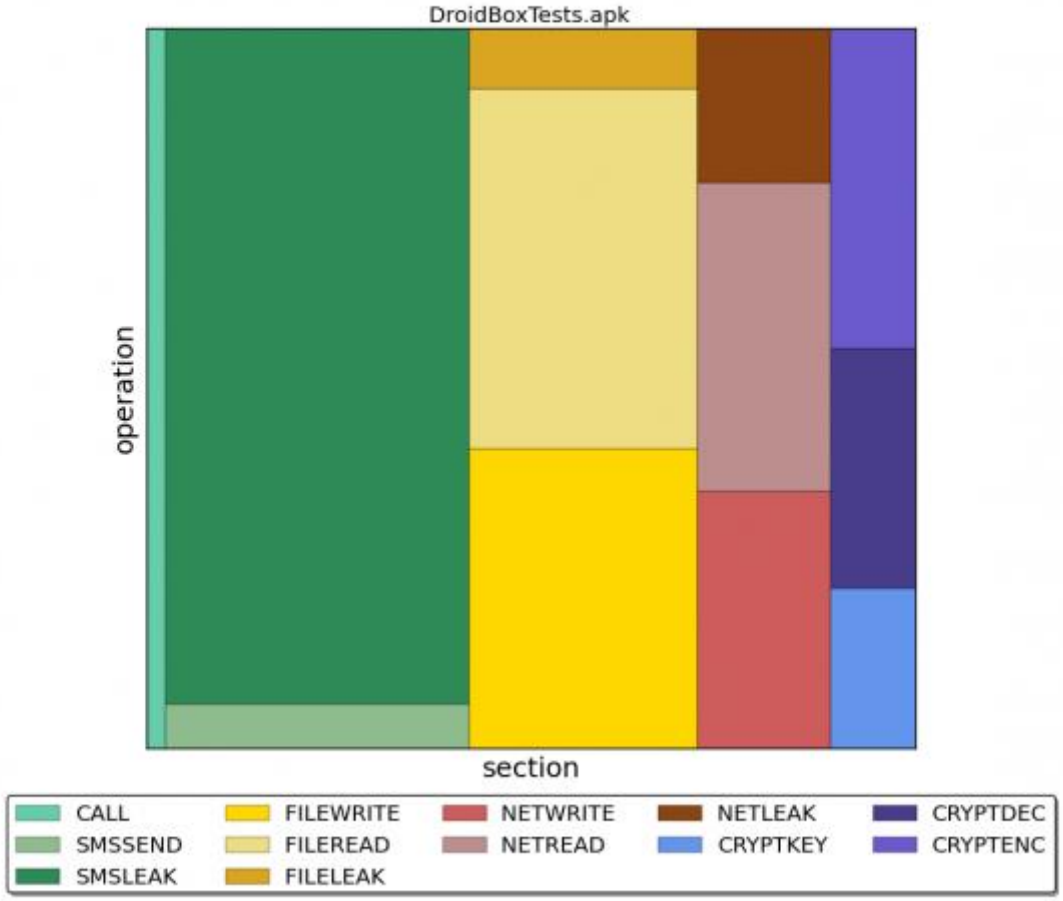

Рисунок 1.3 – Карта работы приложения

В открытом доступе доступны средства для проверки приложений на предмет вредоносности, но они выполняют конкретный функционал и приходится использовать целый список таких приложений, но на данный момент нет приложения, которое бы позволяло использовать весь описанный функционал. Для качественной проверки программ требуется комплексное средство, которое будет сочетать в себе лучшие качества каждого приложения. В то же время в него должно быть включен модуль, позволяющий отображать график состояний ПО, чтобы можно было отслеживать путь вызовов к подозрительному действию и эффективно оценивать схему работы программы.

2 Описание программного обеспечения

2.1 Структура реализованного приложения

Система построения графа состояний приложения включает в себя следующие этапы:

1. Преобразование исходного apk файла в файл jar с использованием библиотеки dex2jar;
2. Получение из jar архива его составляющих – java файлов для каждого класса приложения;
3. Рекурсивный проход по всем java файлам и анализ их содержимого на предмет наличия реализаций методов и их вызовов. Реализация метода отличается от его объявления и вызова наличием фигурных скобок после списка параметров.
4. Для каждого метода определить название и количество переменных. Это будут уникальные свойства, позволяющие отличить одно состояние от другого в графе.
5. В каждой реализации методов найти вызовы других методов, это будут переходы из одного состояния в другое.
6. Записать полученные переходы в файл с форматом .dot, сформировать bat-файл с командой для программы Graphviz и запустить его.
7. После завершения построения Graphviz’ом графа состояний и его экспортом в файл изображения загрузить его и отобразить на форме (в случае с экспортом графа всех состояний изображение на форме не отображается вследствие большого размера).

2.2 Применяемые паттерны

При написании приложения было использовано пять паттернов, которые направлены на удобство поддержания и расширения системы в дальнейшем.

Используемые паттерны:

- *Одиночка*, шаблон проектирования, который предоставляет такую систему взаимодействия с объектами, чтобы у класса существовал только один

экземпляр, к которому нужно создать глобальную точку доступа. Используется для класса ClassAnalizier, чтобы всегда иметь доступ к объекту семантического анализатора. Диаграмма шаблона представлена на рисунке 2.1.

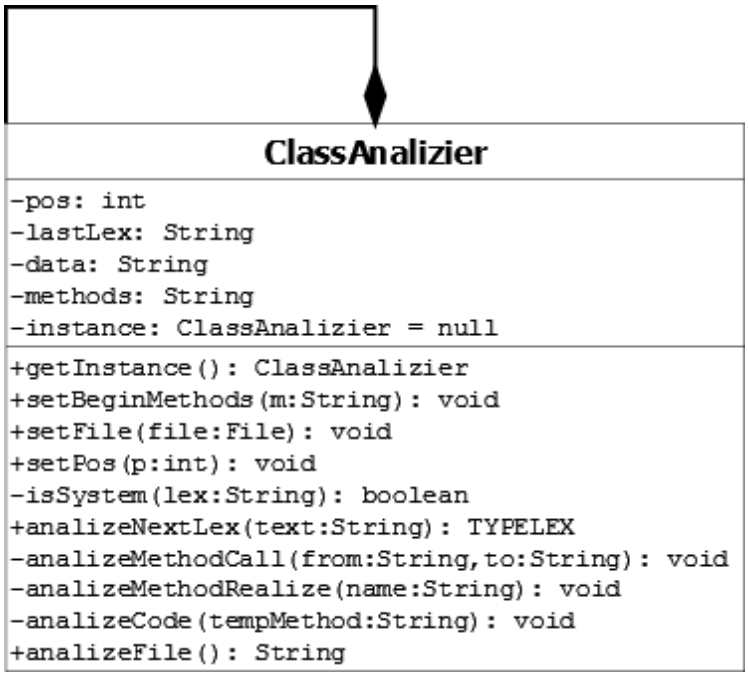

Рисунок 2.1 – Паттерн синглтон

- *Строитель,* который позволяет создавать сложные объекты пошагово. Строитель даёт возможность использовать один и тот же код строительства для получения разных представлений объектов. Строитель представлен интерфейсом AnalizierBuilder, позволяющим определить принципы взаимодействия с разными строителями анализаторов с интерфейсом FileAnalizier. В данном случае представлено два строителя – для ClassAnalizier и ProxyClassAnalizier, имеющие различия в построении и конфигурации объектов. ClassAnalizierBuilder используется в прокси-классе для анализатора, а ProxyAnalizierBuilder – в основном классе управления графическим представлением программы. UML-диаграмма паттерна отображена на рисунке 2.2.

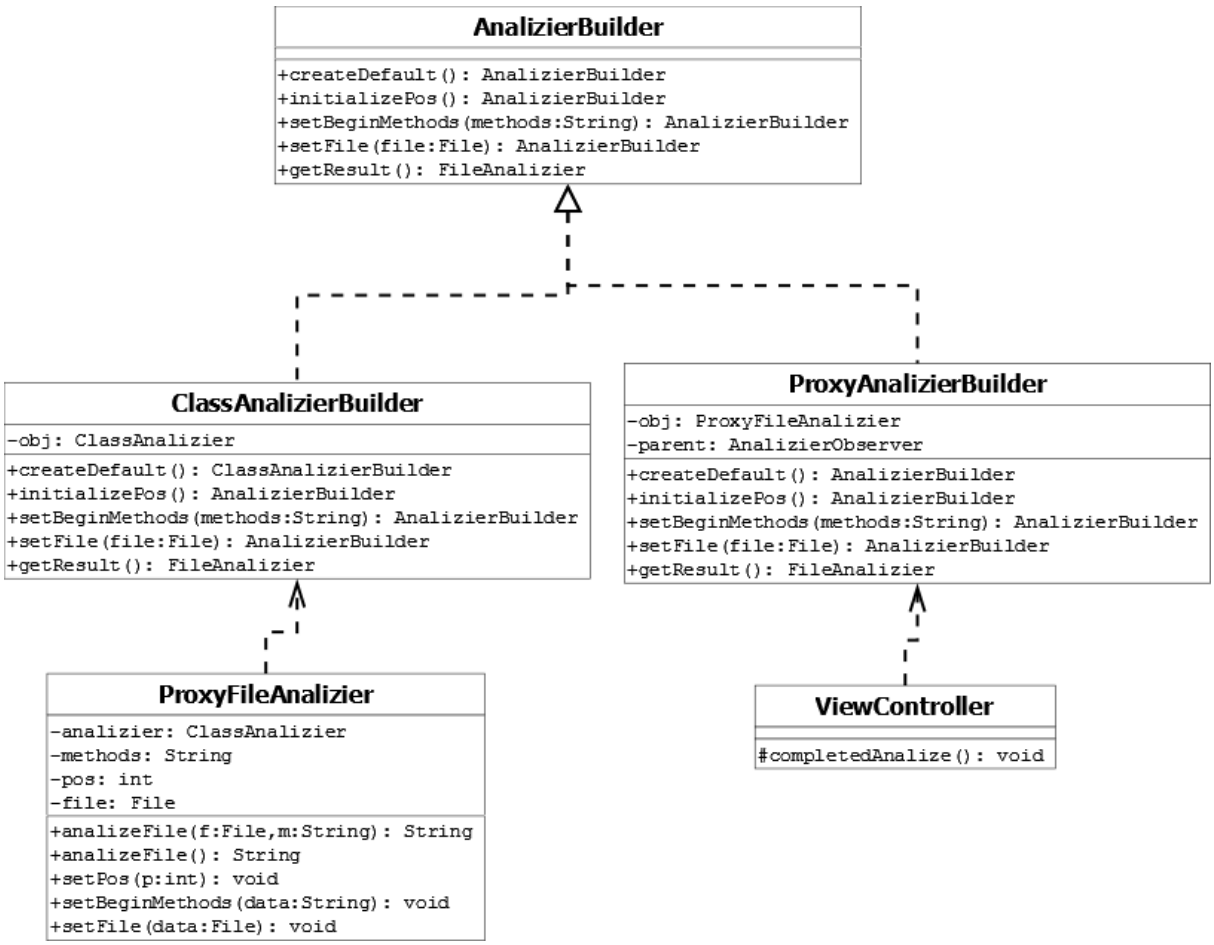

Рисунок 2.2 – Паттерн строитель

- *Делегирование,* передача обязанностей другому классу, который предназначен для решения подобных задач. ProxyFileAnalizier делегирует полномочия классу ClassAnalizier при вызове метода синтаксического анализа файла. Диаграмма этого шаблона показана на рисунке 2.3.

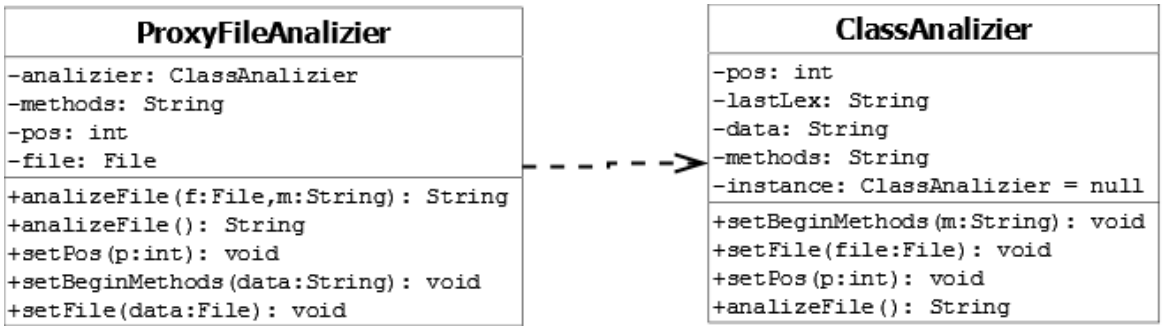

Рисунок 2.3 – Паттерн делегирование

- *Наблюдатель*, — это поведенческий паттерн проектирования, который создаёт механизм оповещения, позволяющий одним объектам следить и реагировать на события, происходящие в других объектах. В данном случае интерфейс оповещателя предполагает, что по завершении текущей задачи он уведомляет наблюдателя и если тот ожидает завершения (что определяется установкой внутреннего флага), то он отреагирует на оповещение вызовом функции обработки. Интерфейс наблюдателя наследует класс управления формой для заполнения progressBar'а в процессе анализа файлов. Оповещателем является ProxyFileAnalizier, который уведомляет о завершении анализа конкретного файла. Наблюдатель отображен на рисунке 2.4.

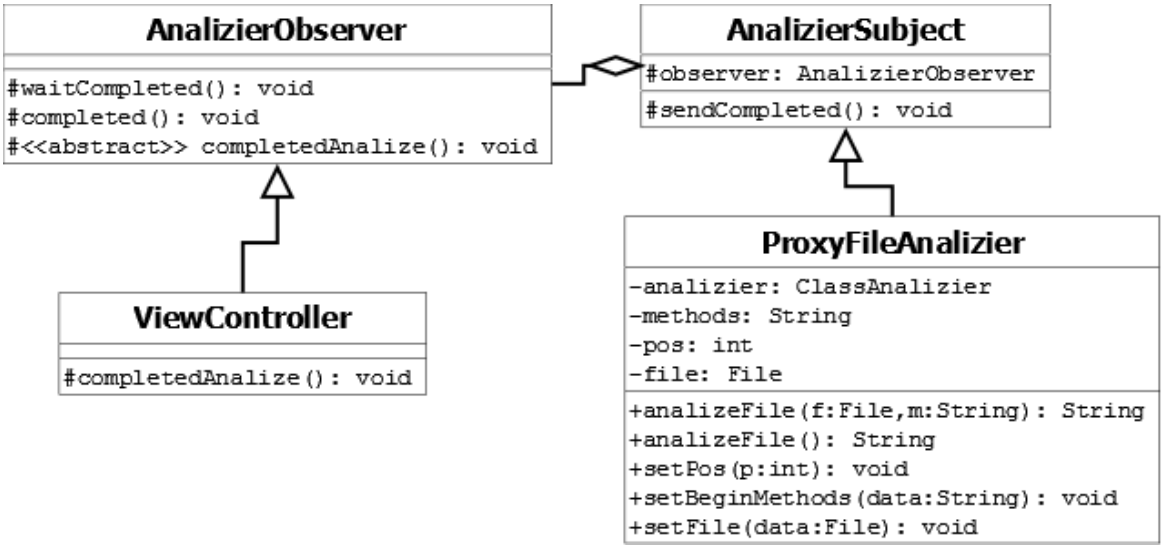

Рисунок 2.4 – Паттерн наблюдатель

- *Заместитель*, структурный шаблон проектирования, предоставляющий объект, который контролирует доступ к другому объекту, перехватывая все вызовы (выполняет функцию контейнера).
  В данном случае заместитель создается для синтаксического анализатора. Если анализатор еще не был закреплен для заместителя, то он получается из статического метода singlton-класса ClassAnalizier и производится его

начальная конфигурация при помощи строителя ClassAnalizierBuilder. При этом, такая проверка проводится только при вызове непосредственно метода обработки файла, при вызове других методов ресурсоемкий объект класса анализатора не создается, а данные накапливаются у заместителя и будут переданы строителю при конфигурации анализатора. На рисунке 2.5 показан паттерн заместитель.

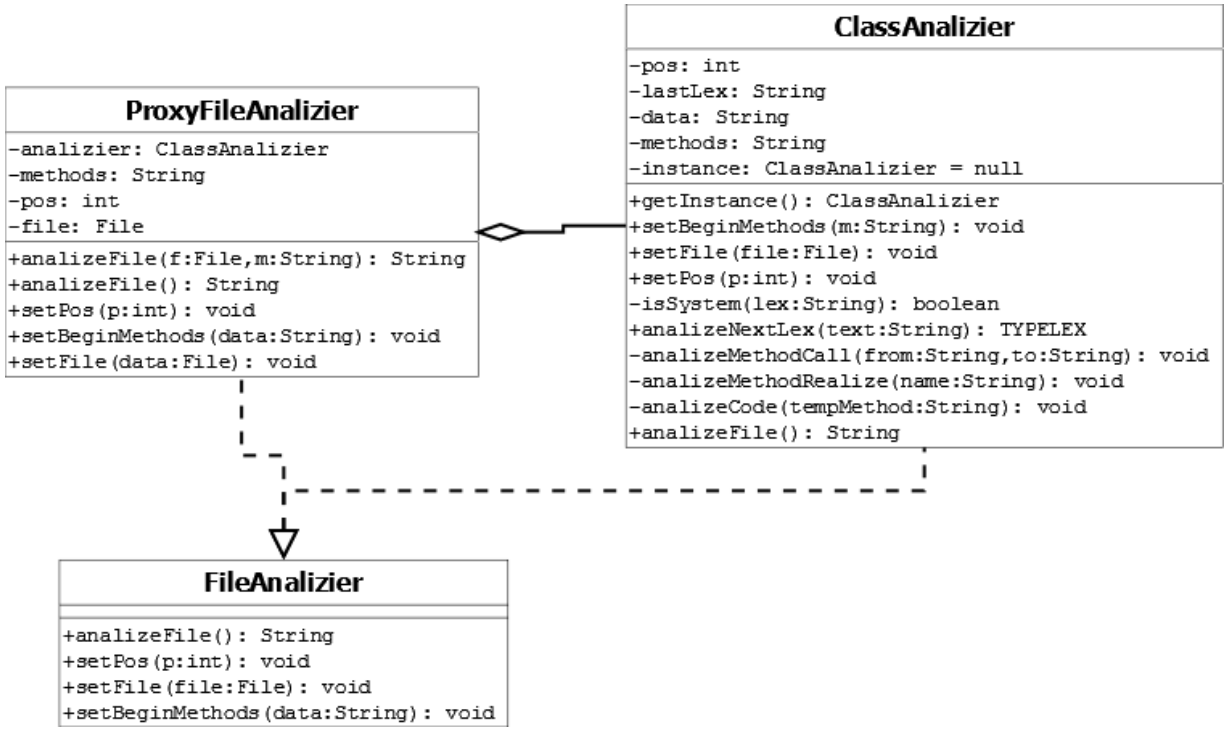

Рисунок 2.5 – Паттерн заместитель

2.3 Диаграмма классов

На рисунке 2.3 отображена полная диаграмма классов программного модуля.

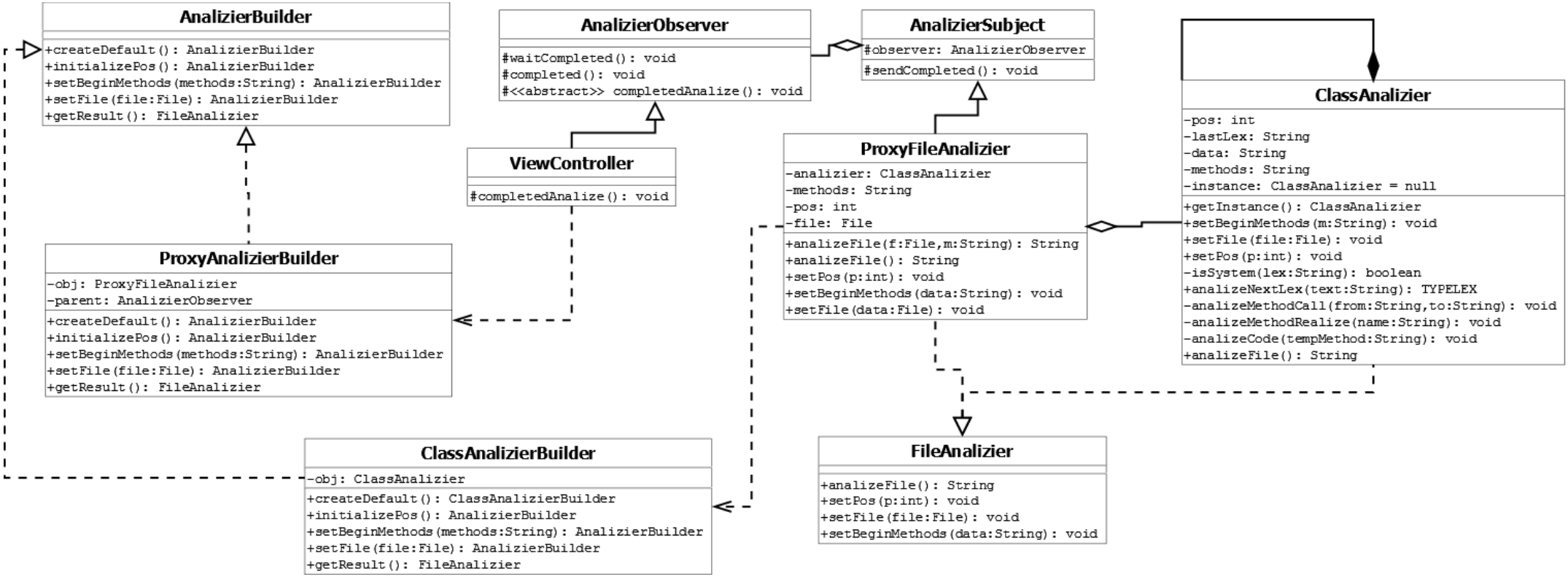

Рисунок 2.3 – Диаграмма классов

2.4 Дальнейшее развитие системы

В рамках использования системы для конкретных задач в реальном мире потребуется добавление модуля семантического анализа к программе для определения классов, обладающих соответствующими методами, а также типов используемых ими данных.

Для анализа методов недостаточно просто видеть их название и количество передаваемых параметров, поэтому стоит для их содержимого проводить анализ воздействия на окружение – запись в файлы, изменение системной конфигурации и указывать эти воздействия на диаграмме состояний, например подсвечивая состояние определенным цветом. Также для удобства подробного рассмотрения состояния следует показывать исходный код состояния при его выборе.

Последующей модификацией стоит добавить определение используемых каждым состоянием прав приложения, например, на запись в файл, отправку сообщений или обращение к интернету. Это позволит выделить ключевые точки взаимодействия программы с внешней средой и определить ее характер.

3 Результаты вычислительного эксперимента

При переходе на вкладку декомпиляции проводится преобразование выбранного на главной вкладке файла apk в формат jar, после чего выводится сообщение об успешном, либо неуспешном завершении (рисунок 3.1).

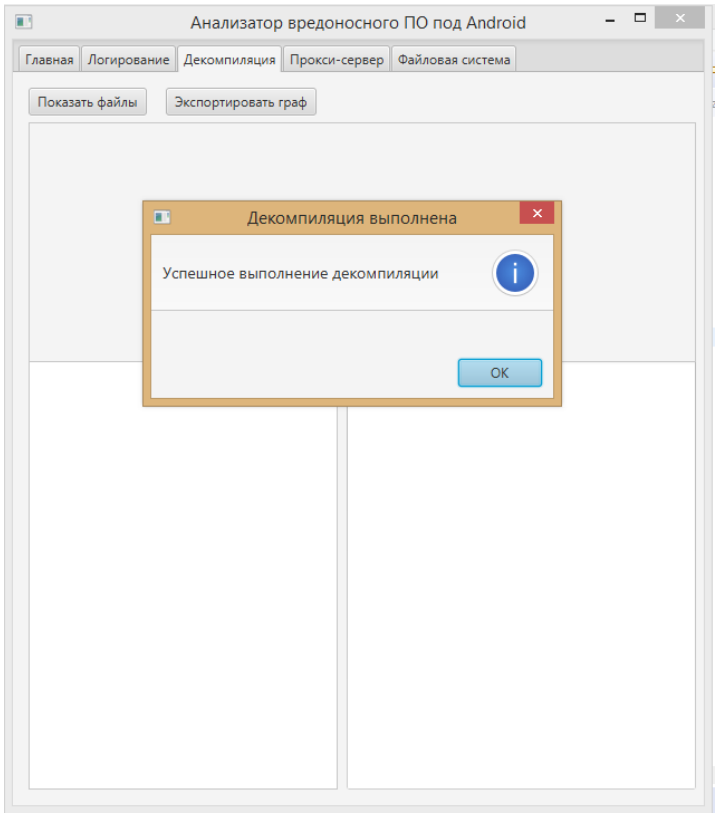

Рисунок 3.1 – Сообщение после декомпиляции

При нажатии на кнопку «Показать файлы» запускается процесс разбора jar архива на java файлы. При этом количество разобранных файлов java отображается на progressBar’е по мере выполнения (рисунок 3.2.). ProgressBar появляется сразу после нажатия на кнопку и исчезает после завершения процесса. Также по его завершении будет выведено дерево файлов приложения, как на рисунке 3.3.

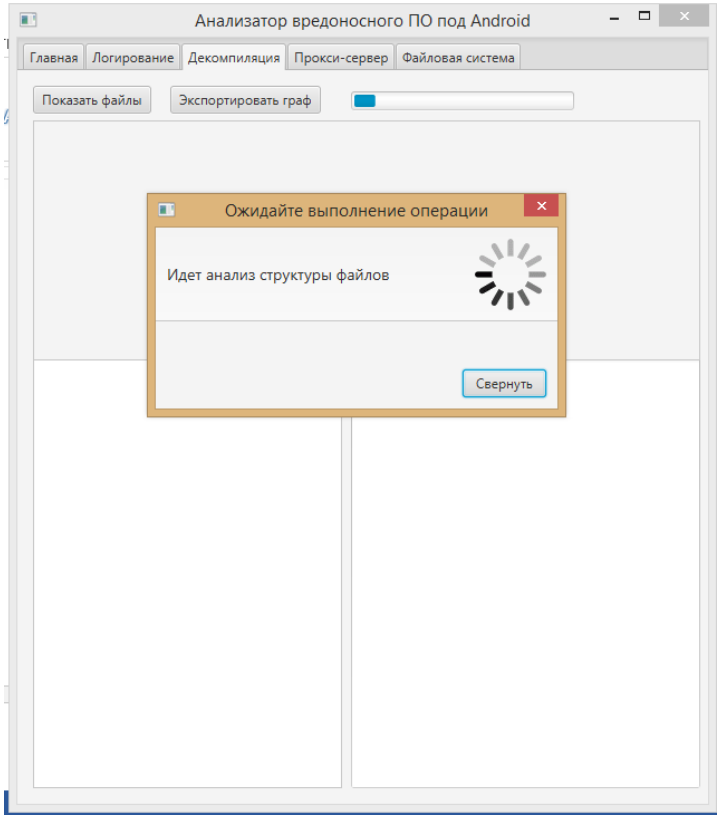

Рисунок 3.2 – Процесс разбора jar на java

При выборе файла в выведенном дереве файлов отображается содержимое этого файла и строится граф его состояний, основанный на коде приложения (рисунок 3.4). Граф можно масштабировать кнопками клавиатуры + и – соответственно, что отображено на рисунке 3.5. На нем отображены методы, реализованные в классе и используемые ими обращения к другим методам.

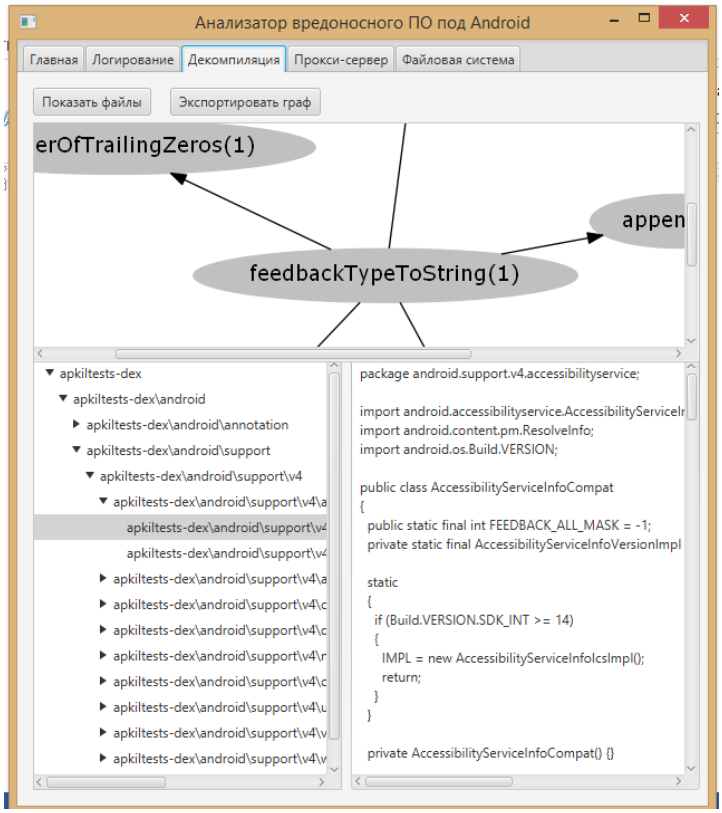

Рисунок 3.3 – Отображение содержимого java-файла

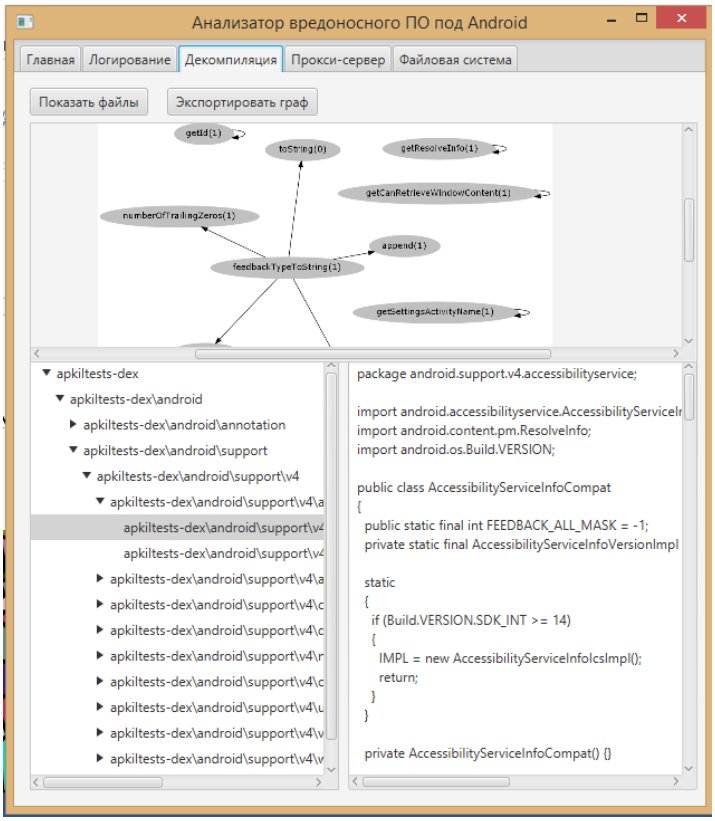

Рисунок 3.4 – Масштабирование графа состояний

Также изображение графа файла экспортируется в директорию программы в файл output.png, который можно открыть и просмотреть (рисунок 3.5).

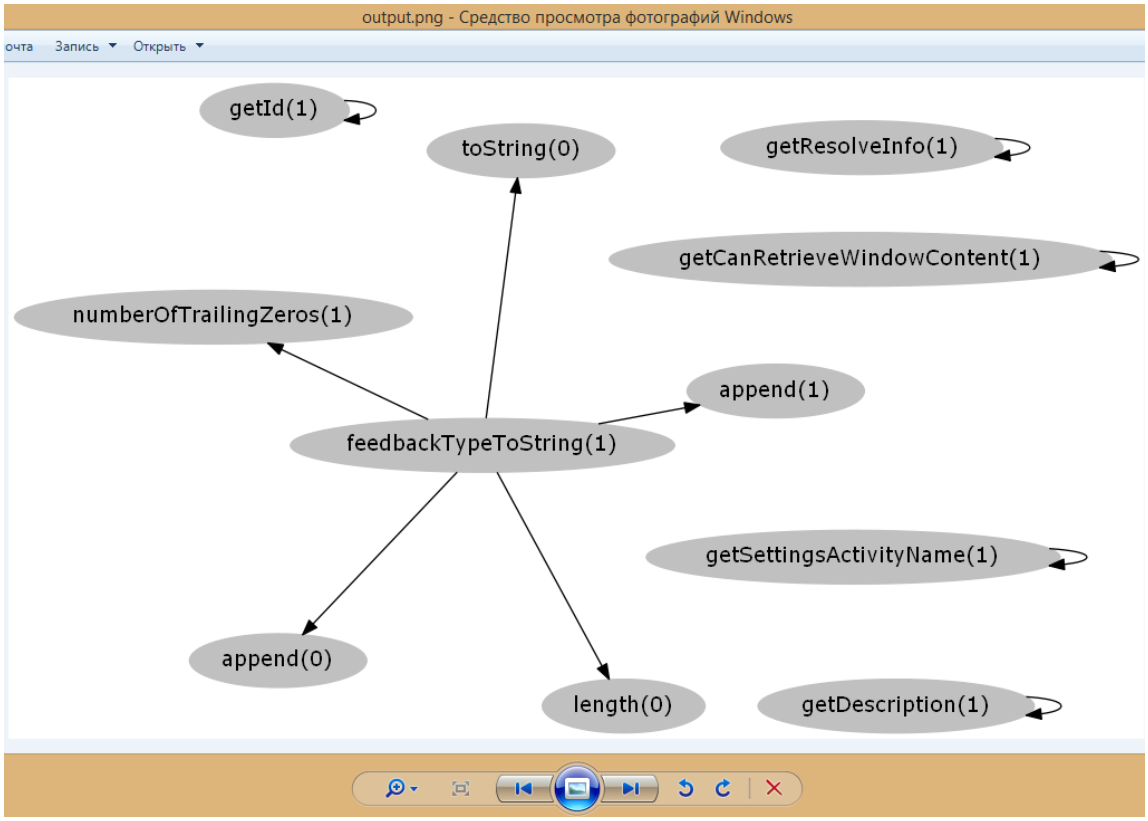

Рисунок 3.5 – Открытый файл output.png

При нажатии «Экспортировать граф» будет построен граф состояний для методов всех файлов и откроется диалоговое окно (рисунок 3.7), в котором можно будет его сохранить. После этого можно найти в выбранном месте этот файл и открыть его для просмотра (рисунок 3.6).

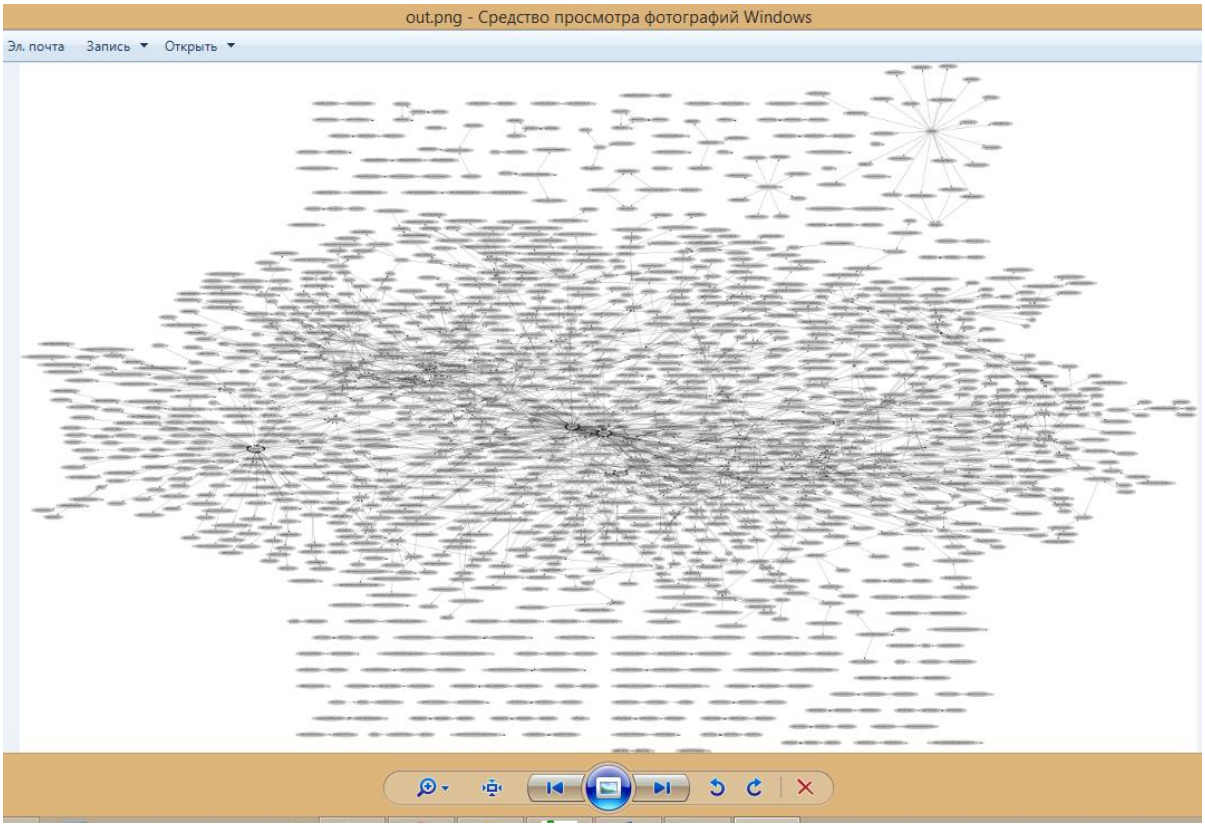

Рисунок 3.6 – Граф всех состояний приложения

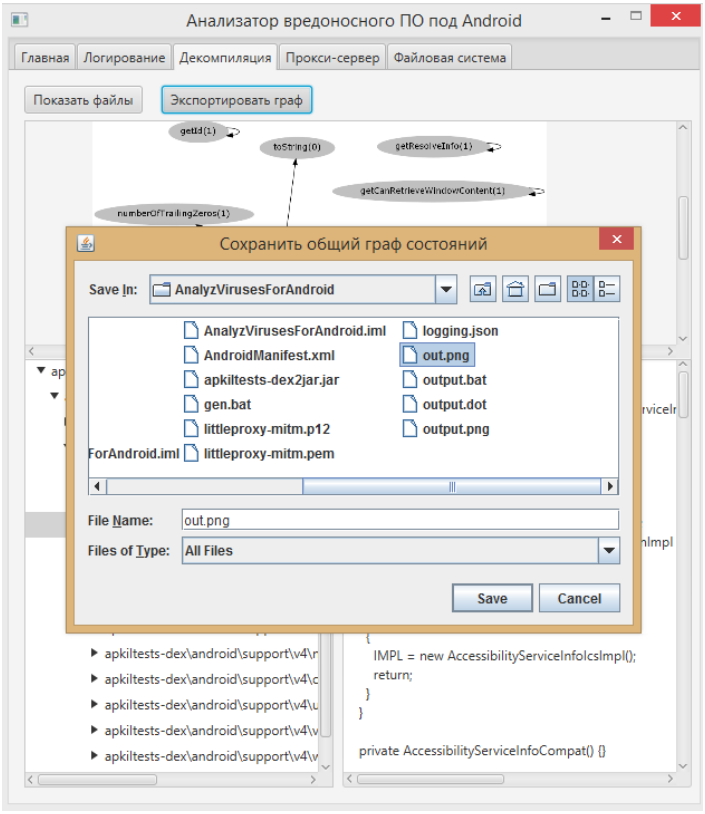

Рисунок 3.7 – Диалоговое окно сохранения файла

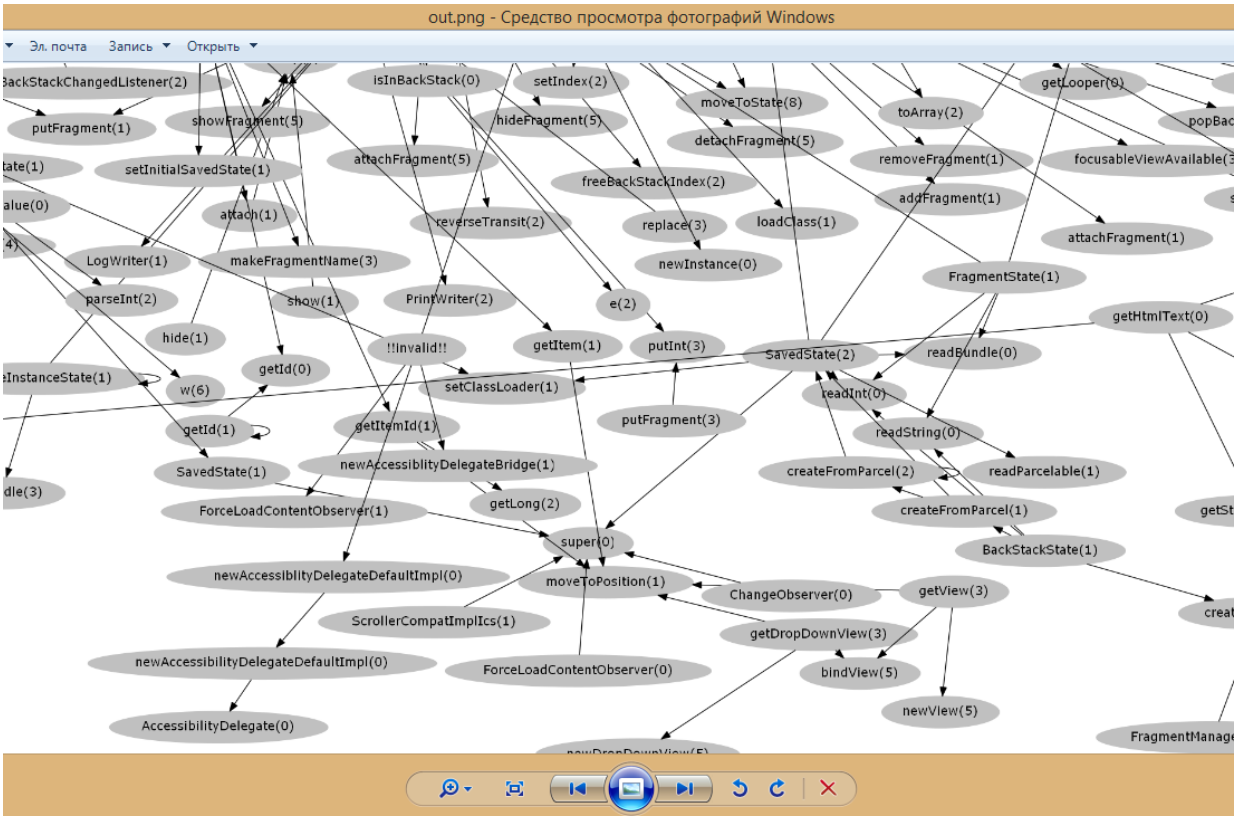

Рисунок 3.8 – Увеличенный фрагмент изображения графа

Заключение

Таким образом, создан модуль приложения, который в автоматическом режиме анализирует apk-приложение, декомпилирует его, используя библиотеку dex2jar и разбирает на множество файлов java. Вследствие чего, можно посмотреть декомпилированный код приложения, а также граф состояний и переходов, отображающий используемые методы каждого файла, либо приложения в целом. Это позволяет понять логику работы приложения.

ПО написано с использованием основных шаблонов проектирования, что предполагает его возможную расширяемость и развитие в дальнейшем путем добавления новых классов в архитектуру.

Список использованных источников

1. Исследование андроид вируса/Хабр [Электронный ресурс]. – URL: https://habr.com/post/255417
2. Вирусы, статистика и немного всего/Geektimes [Электронный ресурс]. – URL: https://geektimes.com/post/293105
3. Троян Loapi для Android – Блог Лаборатории Касперского [Электронный ресурс]. – URL: https://www.kaspersky.ru/blog/loapi-trojan/19382
4. DJ Java Decompiler – java disassembler, decompiler and editor [Электронный ресурс]. – URL: http://www.neshkov.com/dj.html
5. Graphviz – Graph Visualization Software [Электронный ресурс]. – URL: https://www.graphviz.org/
6. GitHub – pjlantz/droidbox: Dynamic analysis of Android apps [Электронный ресурс]. – URL: https://github.com/pjlantz/droidbox

Приложение А. Код программы

Исходный код разработанной программы находится на прилагаемом диске, в соответствующем каталоге.