УНИВЕРСИТЕТ ИТМО

ФГАОУ ВО «Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

ПРОЕКТИРОВАНИЕ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Лабораторная работа №2

Проектирование архитектуры вычислительной системы

Лабушев Тимофей

Группа P3402

Санкт-Петербург

2020

Анализ архитектурных проблем и доступных средств проектирования

1. **Обеспечение совместимости с различными наборами команд**

   Архитектурный стиль, инструменты проектирования: потоки данных (data flow)

   Преобразование входных данных в промежуточное представление, с которым работают анализаторы, удобнее всего представить как конвейер. Поскольку на данном этапе может осуществляться взаимодействие с внешними программами, целесообразно продумать не только используемые компоненты (*filters*), но и то, как они друг с другом соединяются и как между ними передаются данные (*pipes*).

2. **Предоставление выходного отчета в различных форматах**

   Архитектурный стиль, инструменты проектирования: диаграммы прецедентов

   Для того, чтобы сформулировать требования к выходным данным, необходимо изучить различные способы взаимодействия пользователя с системой — через интерфейс командной строки, через расширение среды разработки и т.д.

3. **Разрешение зависимостей между анализаторами**

   Архитектурный стиль, инструменты проектирования: диаграммы состояния, деятельности

   Требуется рассмотреть все состояния, в которых может находится система в процессе выполнения каждого анализатора, а также решения, влияющие на переходы между ними. Для этой задачи подходит описание как в виде диаграмм состояния, так и в виде диаграмм деятельности.

4. **Возможность параллельного выполнения анализаторов**

   Архитектурный стиль, инструменты проектирования: диаграммы последовательности

   Позволяют рассмотреть взаимодействие исполнителей и контроллера исполнения. Описанная выше проблема разрешения зависимостей требует определенного механизма синхронизации при параллельном исполнении анализаторов, варианты реализации которого можно сравнить именно с помощью диаграмм последовательности.

5. **Динамическая конфигурация анализаторов**

   Архитектурный стиль, инструменты проектирования: диаграммы последовательности

   Проблему динамической конфигурации можно свести к выбору между построением библиотеки или фреймворка. Диаграмма последовательности иллюстрирует порядок взаимодействия между пользовательским приложением, анализаторами и компонентами системы, позволяя решить, насколько оправдан тот или иной подход.

Рассмотрение архитектурных решений

Обеспечение совместимости с различными наборами команд

Получение промежуточного представления, над которым производится анализ, возможно несколькими способами:

1. *(Рис. 1)* Исходный файл обрабатывается внешним дизассемблером (например, `llvm-objdump`). На вход системе поступает ассемблерный листинг с метаданными, который считывается парсером.
2. *(Рис. 2)* Система обрабатывает исходный файл с помощью библиотеки `LLVM`, считывая по одной инструкции за шаг. Не требуется парсер текстового листинга, но необходима обработка формата ELF.

Первый вариант реализации позволяет использовать альтернативные дизассемблеры или выполнять определенные преобразования листинга до анализа. Помимо этого, система упрощается за счет отсутствия интегрированных зависимостей.

Тем не менее, приведенные ниже диаграммы показывают, что оптимальным подходом является интеграция дизассемблера и бинарного парсера. Во-первых, это позволяет извлекать двоичные метаданные в промежуточные структуры без необходимости их сериализации и десериализации. Во-вторых, уменьшаются вариации во входных данных, что облегчает верификацию системы.

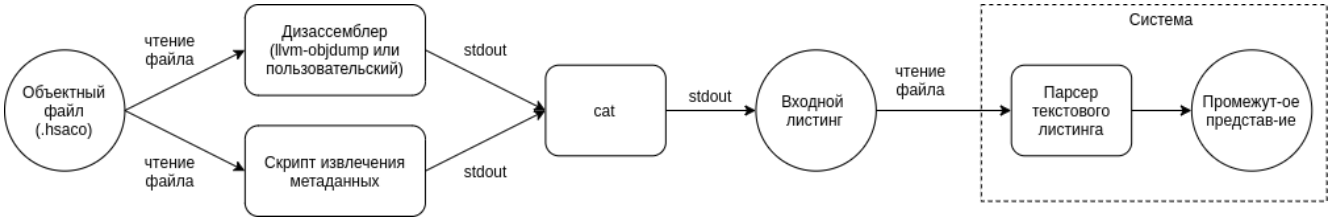

Рис. 1: Обеспечение совместимости с различными наборами команд. Использование внешнего дизассемблера.

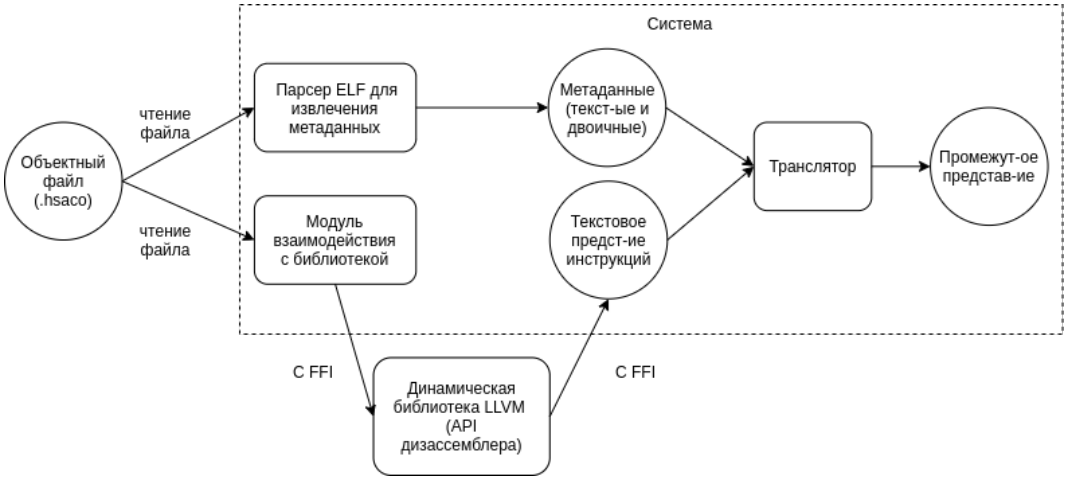

Рис. 2: Обеспечение совместимости с различными наборами команд. Интеграция дизассемблера.

Разрешение зависимостей между анализаторами

Выходное значение одного анализатора может быть входным значением другого. За разрешение зависимостей отвечает контекст исполнения. Были рассмотрены следующие варианты его реализации:

1. *(Рис. 3)* Анализатор имеет доступ к контексту и может в любой момент исполнения запросить зависимость.
2. *(Рис. 4)* Зависимости разрешаются перед исполнением анализаторов и передаются как входные аргументы.

Первый вариант является более простым в реализации и позволяет динамически определять зависимости — они могут различаться в зависимости от семейства ГП или каких-либо внешних настроек.

Второй подход, выбранный в качестве решения проблемы, является менее гибким, но обладает существенным преимуществом: он позволяет избежать синхронизации во время исполнения, что важно для параллелизации (проблема 4) и наглядно иллюстрируется выбранным инструментом проектирования — диаграммами деятельности.

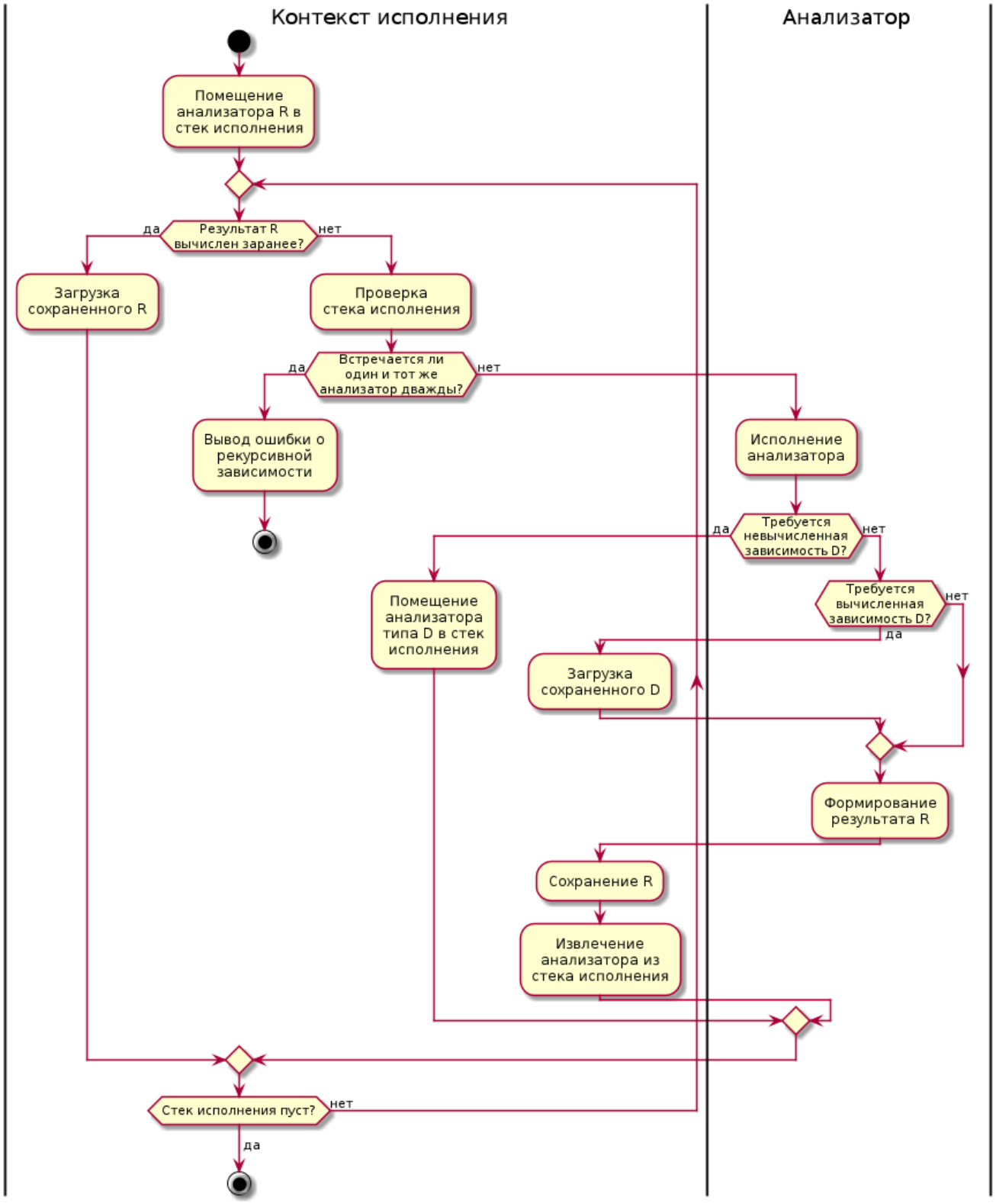

Рис. 3: Разрешение зависимостей между анализаторами во время исполнения.

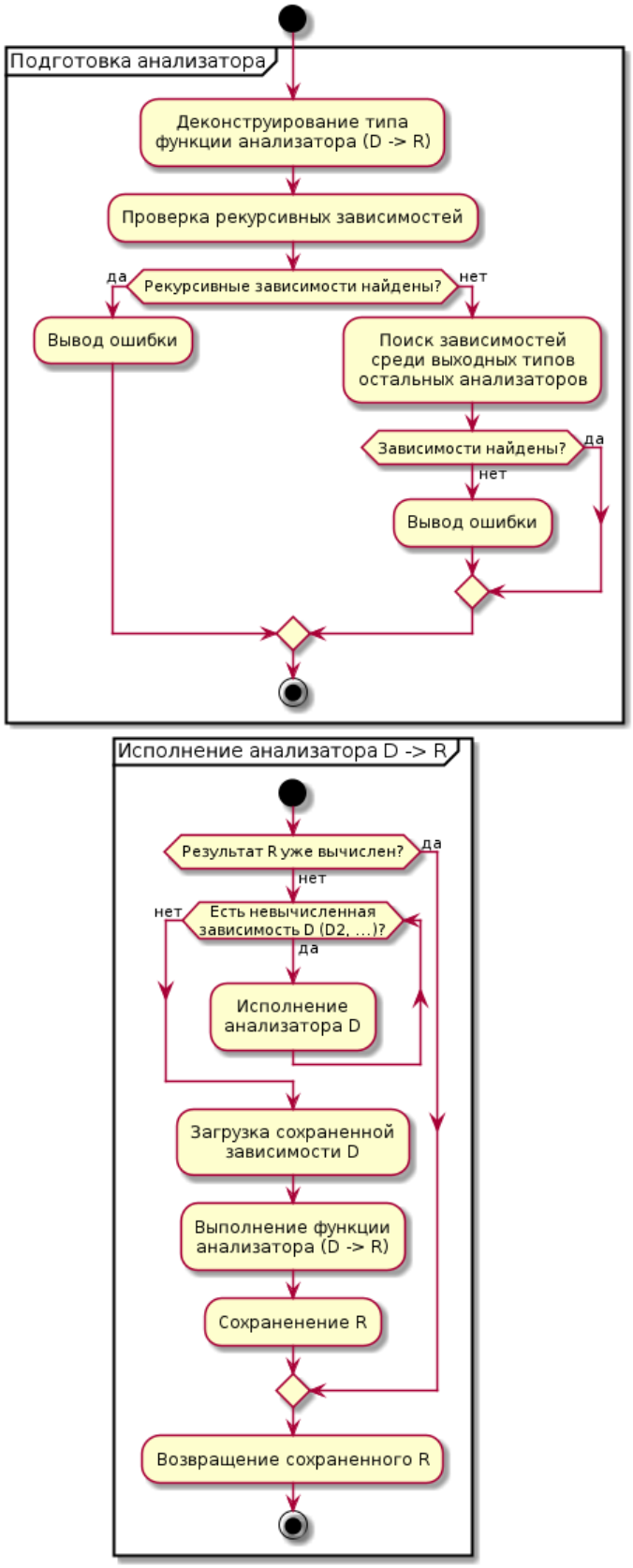

Рис. 4: Разрешение зависимостей между анализаторами перед исполнением.

Динамическая конфигурация анализаторов

Пользовательское приложение состоит из анализаторов и «склеивающего» кода, обеспечивающего их запуск. Передача параметров анализаторам и определение, какие из них подходят для текущей конфигурации, возможно следующими путями:

1. *(Рис. 5)* Каждый анализатор перед выполнением сообщает о своей применимости, основываясь на доступных ему метаданных программы и внешнем окружении (файлах конфигурации, переменных среды).
2. *(Рис. 6)* Каждый шаг анализа — дизассемблирование, конфигурация анализаторов и т.д. — контролируется пользовательским приложением. Контекст исполнения создается уже после выбора подходящих анализаторов.

Первый путь позволяет максимально сократить количество «склеивающего» кода, поскольку все компоненты системы инкапсулированы в контексте, а также делает функции анализа более независимыми, что упрощает их переиспользование.

Второй подход, выбранный в качестве решения, немного увеличивает время, затрачиваемое пользователем на интеграцию анализаторов, но при этом предоставляет большую гибкость в работе с отдельными компонентами, что демонстрируется диаграммами последовательности, на которых пользовательское приложение изображено одним из участников процесса.

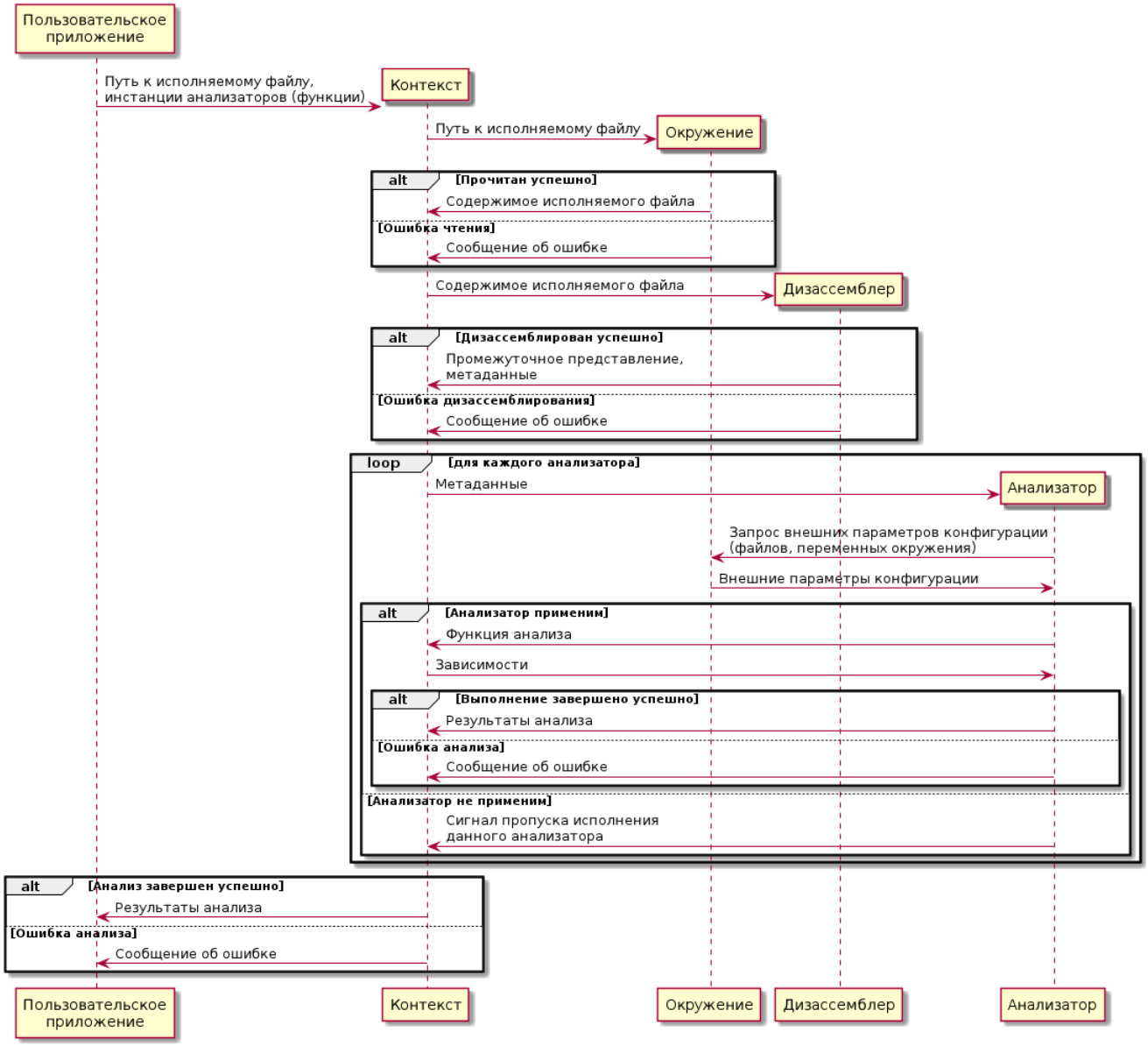

Рис. 5: Динамическая конфигурация анализаторов. Контекст и анализаторы независимы.

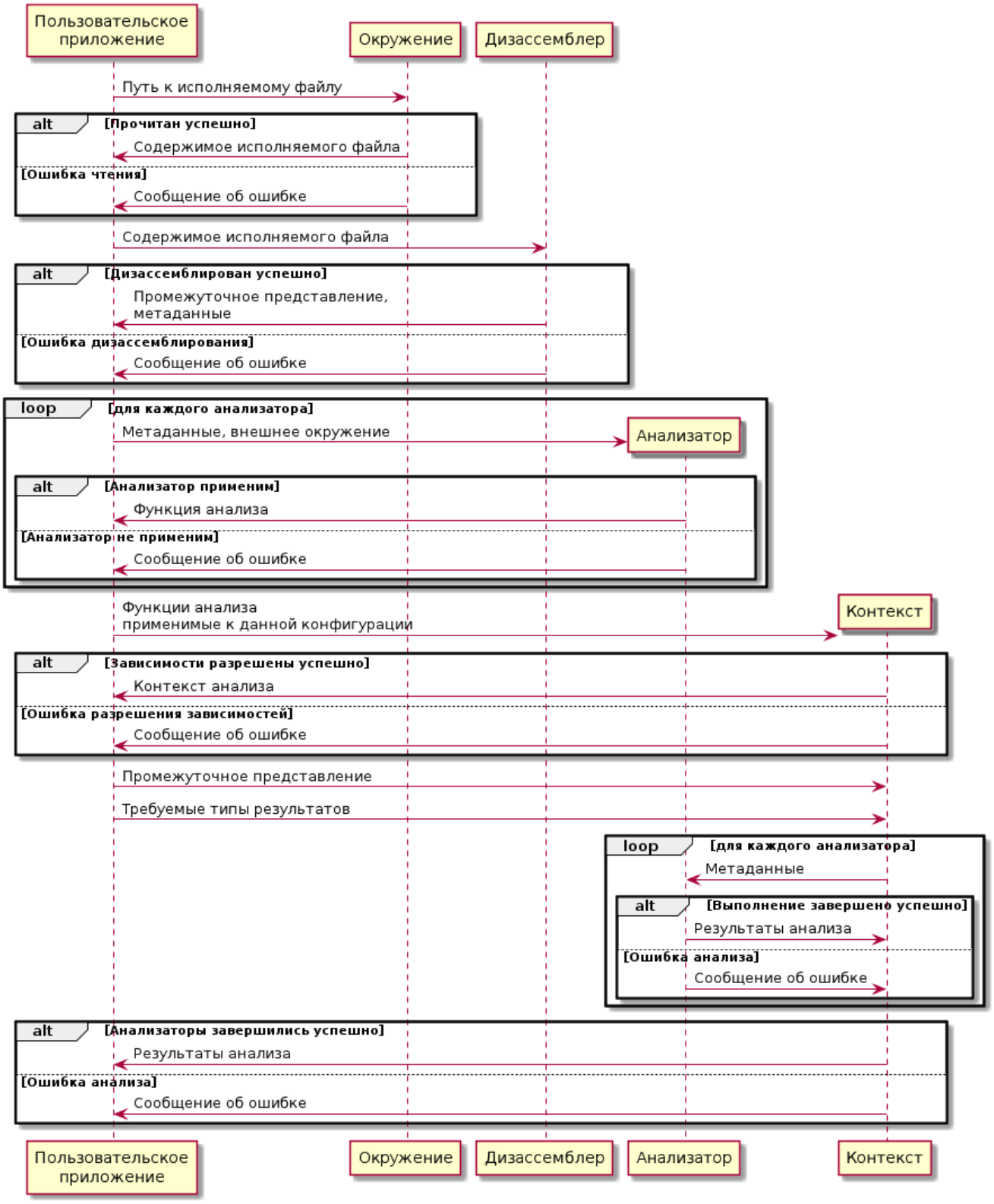

Рис. 6: Динамическая конфигурация анализаторов. Приложение контролирует контекст и анализаторы.

Вывод

В ходе выполнения лабораторной работы были приобретены навыки использования инструментов архитектурного проектирования в решении конкретных архитектурных проблем. Было проведено сравнение альтернативных подходов при помощи различных видов диаграмм, что позволило в каждом случае выбрать оптимальное решение.