Национальный исследовательский университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №2

по дисциплине Проектирование вычислительных систем

Студент: Саржевский Иван

Группа: P3402

г. Санкт-Петербург

2020 г.

Цель

Получить навыки документирования и проектирования архитектуры вычислительных систем.

Архитектурные проблемы

1. Выбор модели взаимодействия между расширением и сервером.
2. Тип протокола взаимодействия расширения и сервера: `Stateful/Stateless`.
3. Пользовательский интерфейс конфигурации шагов отладки.
4. Выбор оптимального интерфейса представления данных с учетом специфики системы.
5. Поддержка конфигураций нескольких проектов в рамках одного решения.

Анализ доступных инструментов архитектурного проектирования и их применимости к решению выявленных архитектурных проблем исследуемого проекта

Архитектурная проблема	Вид диаграммы	Применимость
Выбор модели взаимодействия между расширением и сервером	Deployment	Развертывания для описания концепта, когда один хост может работать с несколькими удаленными машинами.
Тип протокола взаимодействия расширения и сервера: `Stateful/Stateless`	Activity, Sequence	Деятельности для описания процесса взаимодействия клиента с сервером на уровне базовых команд. Последовательности для описания использования базовых операций для настройки пользовательских сценариев.
Пользовательский интерфейс конфигурации шагов отладки	Use-Case	Для описания набора сценариев, который определяет интерфейс взаимодействия пользователя с системой профилей.
Выбор оптимального интерфейса представления данных с учетом специфики системы	Use-Case	Для описания сценариев работы пользователя с данными, для удобной реализации которых служит интерфейс представления данных.
Поддержка конфигураций нескольких проектов в рамках одного решения	Component	Для иллюстрации работы расширения с внешними физическими компонентами - файлами с конфигурациями проектов, в рамках одного решения.

Документирование архитектурных решений

Пользовательский интерфейс конфигурации шагов отладки

Краткое описание решения: Суть проблемы заключается в том, что, во первых, при разработке шейдера у программиста есть необходимость в автоматизации множества различных сценариев взаимодействия (запуск отладчика, дизассемблера, профайлера), и во вторых, есть потребность в запуске этих сценариев на различных целевых платформах. Соответственно, необходимо было обеспечить достаточную гибкость настройки пользовательских сценариев, при этом явно разграничить настройки окружения для разных целевых платформ, с возможность переключения между ними в один клик.

Решением проблемы стала система Профилей. Профиль определяет рабочее окружение и некоторые стандартные параметры (ip-адресс целевой машины, порт, на котором работает сервер, рабочая директория итд.), конфигурацию для отладки и набор пользовательских сценариев - Action'ов. По сути, отладка - это доступный по умолчанию Action, который интегрирован с интерфейсом отладки VS. Action состоит из набора базовых операций, Step'ов, выполняющихся в определенном порядке (Execute, Copy File, Open in Editor, Run Action).

Схемное решение проблемы: Для описания была использована диаграмма прецедентов (см. рис. 1).

Пояснение и вывод: Диаграмма прецедентов в этом случае отражает отражает ожидаемые сценарии использования системы с точки зрения пользователя.

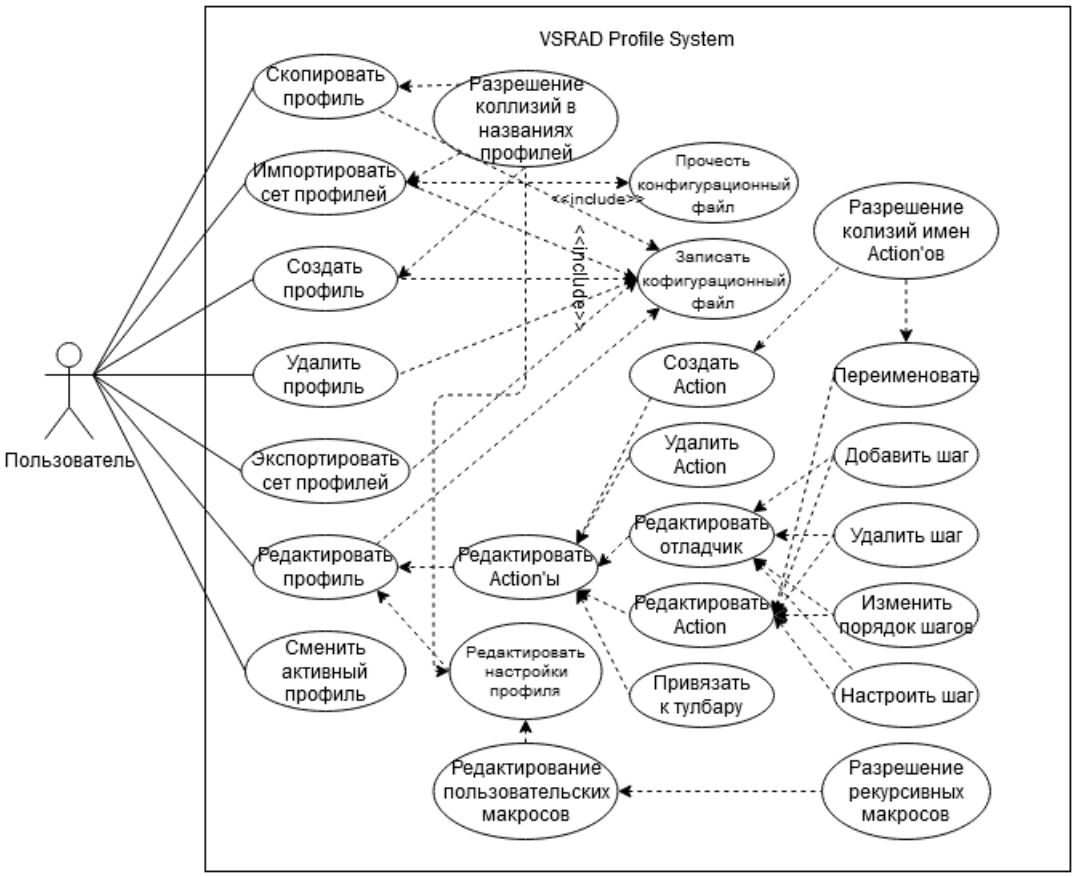

Рис. 1: Диаграмма прецедентов для системы профилей

Выбор модели взаимодействия между расширением и сервером

Краткое описание решения: В угоду упрощения программного продукта, можно запускать механизм отладки непосредственно на машине, на которой ведется разработка. Однако такое решение имеет ряд недостатков с точки зрения особенностей рассматриваемой системы. В частности, отладка шейдера обычно производится на нескольких целевых платформах, в таком случае для организации работы необходимо иметь на каждой целевой платформе Visual Studio (что означает, что все платформы должны быть под управлением OS Windows) с установленным расширением, а все изменения в исходниках необходимо синхронизировать через систему контроля версий или локальный файловый сервер. Клиент-серверная модель решает эту проблему, клиент перед каждым запуском отладки отправляет исходник на целевой хост и при необходимости можно переключиться на другую удаленную машину без необходимости физически переходить на другую машину (или подключать rdp) и вручную синхронизировать исходники. Помимо этого, некорректное поведение шейдера может привести к печальным последствиям, вплоть до необходимости перезагружать машину. Клиент-серверная архитектура позволяет избежать потери несохраненных изменений в таком случае.

Схемное решение проблемы: Для описания были использованы диаграммы развертывания (см. рис. 2 и рис. 3).

Пояснение и вывод: Приведены диаграммы развертывания для двух вариантов решения архитектурной проблемы. Исходя из приведенных диаграмм, достоинства выбранного варианта очевидны.

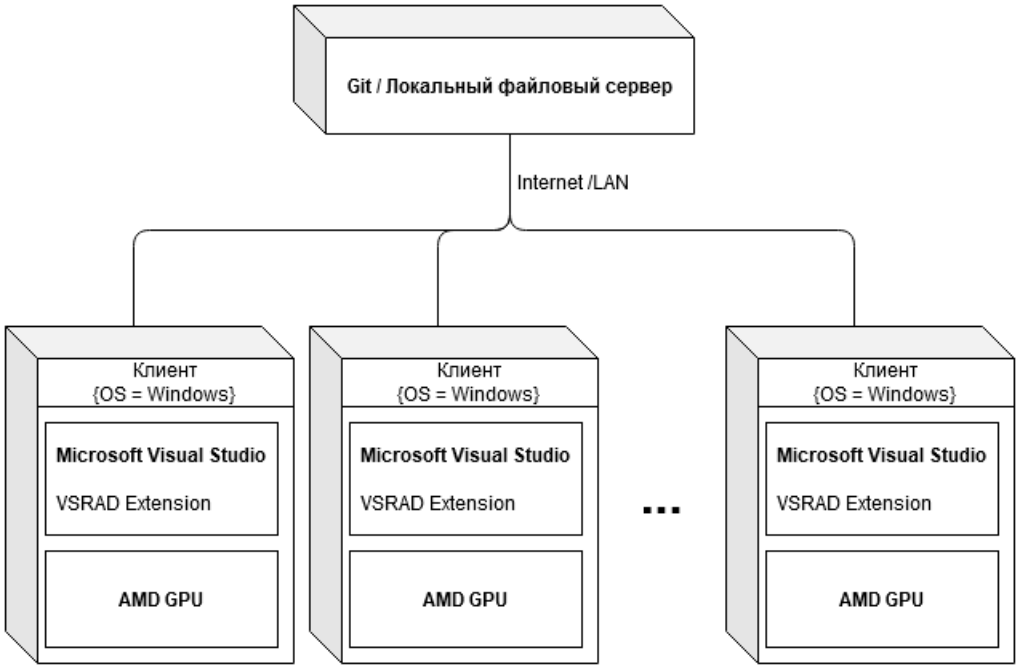

Рис. 2: Диаграмма развертывания для отладки на хосте

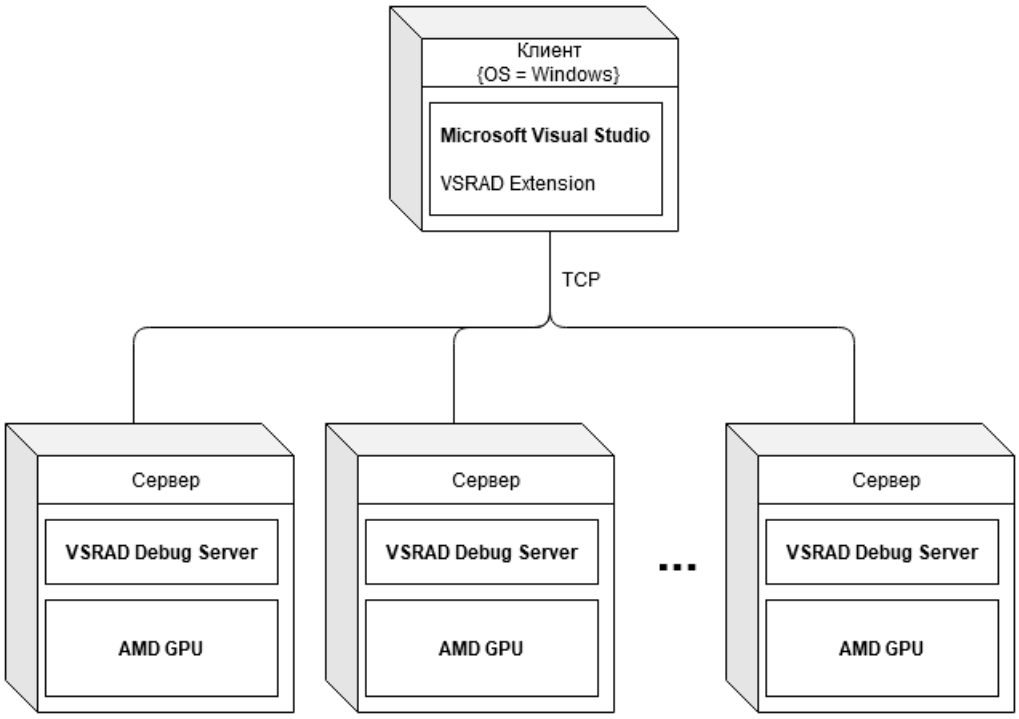

Рис. 3: Диаграмма развертывания для отладки с клиент-серверной архитектурой

Тип протокола взаимодействия расширения и сервера: `Stateful/Stateless`

Краткое описание решения: Изначально хотелось минимизировать количество передаваемой информации между расширением и сервером, и так как сначала была поддержка только отладки, была разработана система, при которой расширение отправляет единственное сообщение серверу и он отвечает одним сообщением. В дальнейшем стали появляться новые сценарии использования (дизассемблирование, профилирование, деплой, билд итд.), соответственно появлялась новая логика обработки этих команд. Стало очевидно, что архитектура, которая требует внесения изменений в код как расширения, так и сервера для добавления нового сценария, нам не подходит.

Было решено выделить во всех сценариях базовые операции и собирать сценарии из этих базовых операций. Таким образом удалось добиться максимальной гибкости настройки расширения и избавиться от необходимости вносить изменения в код для добавления сценариев.

Схемное решение проблемы: Для описания были использованы диаграммы деятельности (см. рис. 4 и рис. 5) и диаграмма последовательности (см. рис. 6).

Пояснение и вывод: Диаграммы деятельности показывают как обрабатываются клиентом и сервером команды в случае с целыми командами на сценарий и командами для базовых операций. Диаграмма последовательности иллюстрирует как можно настроить отладку используя базовые команды.

Вывод

В ходе выполнения данной лабораторной работы были рассмотрены инструменты архитектурного проектирования в контексте решения конкретных архитектурных проблем. Были составлены диаграммы, описывающие принятые архитектурные решения и диаграммы, позволяющие сравнить несколько возможных архитектурных решений.

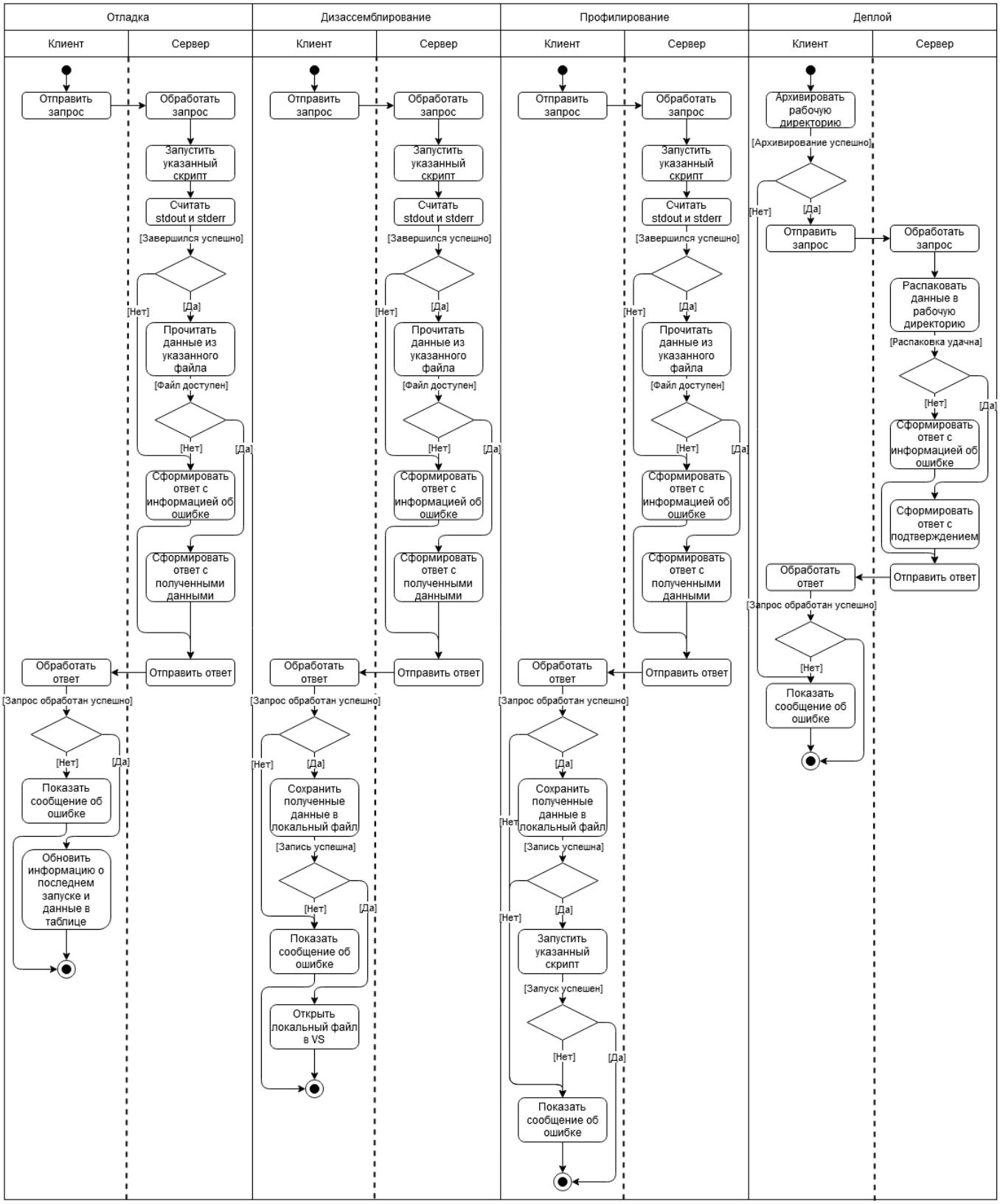

Рис. 4: Диаграмма деятельности, команда на сценарий

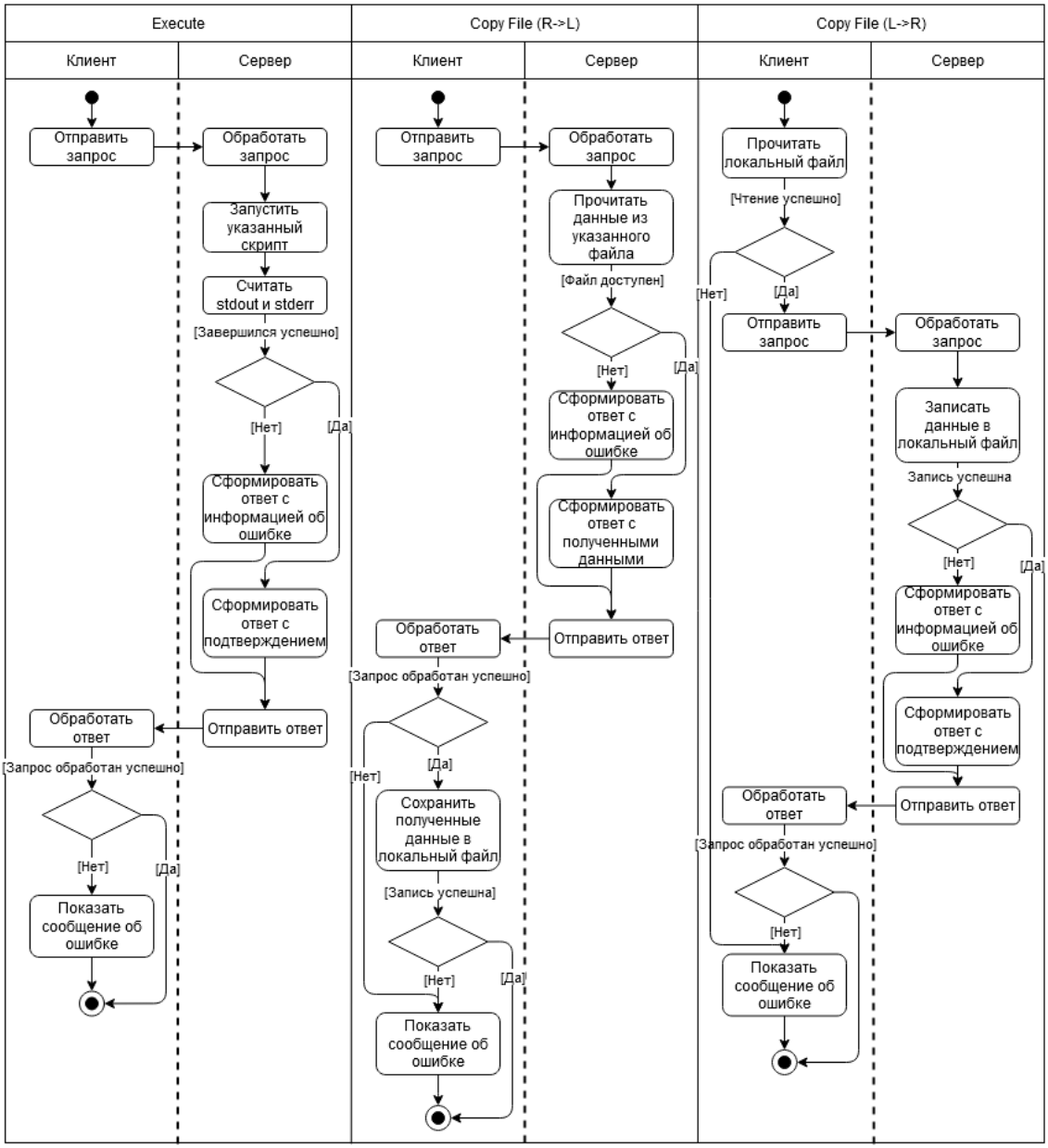

Рис. 5: Диаграмма деятельности, базовые операции

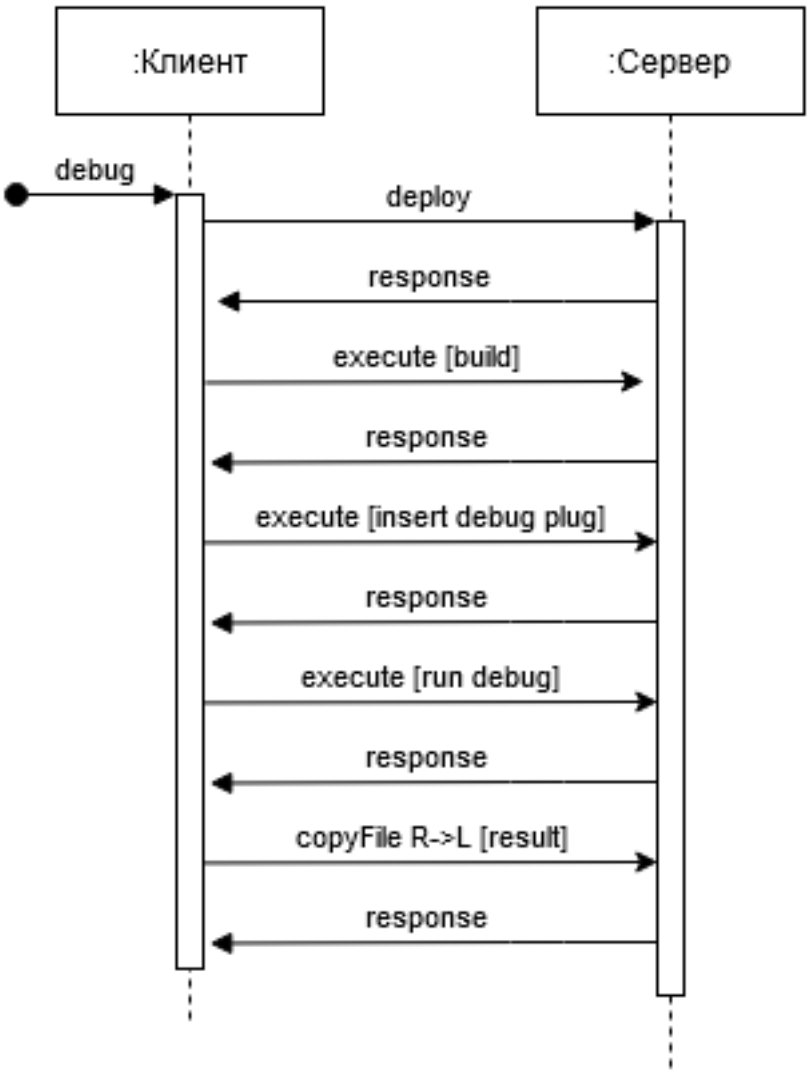

Рис. 6: Диаграмма последовательности, пример отладки с помощью базовых операций