Спецификация диаграммы Эффектов

v0.0.2

История изменений

v0.0.2

1. Отказ от представления событий отдельными блоками
2. Переход на визуальный язык C4 Model

v0.0.1

1. Первая версия

Введение

Диаграмма Эффектов - это инструмент для разработчиков, созданный для визуализации поведения информационной системы через её эффекты. Эффекты - это взаимодействие системы с внешним миром (например, запись и чтение диска, обращение к БД, обращение к внешним сервисам и т.п.).

Такое представление полезно на всех этапах разработки ПО - от первичного анализа до планирования работ и поддержки.

Концептуальная модель

В основе концептуальной модели диаграммы лежат **эффекты** - атомарные единицы взаимодействия системы с окружающим миром. Группы эффектов, которые обеспечивают атомарные функции системы объединяются в **операции**. Элементы окружающего мира, с которыми могут взаимодействовать операции, представлены внутри системы **ресурсами**. Наконец, выполнение операций инициируется **событиями**.

Эффект

Эффект - это акт чтения или изменения некоторого изменяемого состояния. Это может быть состояние внешнего устройство, доступное через операции ввода-вывода, или глобальная память программы, доступная напрямую.

Эффекты чтения характеризуются тем, что считывание одного и того же состояния в разные моменты времени может дать разные результаты. А эффекты изменения (или записи) характеризуются тем, что изменённое состояние можно наблюдать за пределами текущего потока исполнения.

Примерами эффектов являются:

1. чтение и запись статической переменной
2. чтение и запись файла
3. выборка или обновление данных в таблице реляционной БД
4. отправка HTTP GET и POST запросов

Понятие эффекта подробнее рассмотрено в "Приложение 2. Философия эффекта" этого документа.

Операция

Как правило, для реализации функции системы требуется несколько эффектов. Как минимум что-то считать и потом записать обратно. Группы эффектов, реализующих одну функцию системы, образуют операции системы.

Ресурс

У эффектов, которые являются действиями, есть объект - целевое состояние. Это некоторая часть физического мира, с которой будет взаимодействовать устройство ввода-вывода в процессе реализации эффекта. Чаще всего целевым состоянием выступают биты на носителях информации, но это могут быть и пиксели экрана, и динамик колонки и актуатор робота. Эти части физического мира представлены в системе ресурсами.

Событие

Программа выполняет набор эффектов в ответ на событие. Таким событием может быть получение вызова удалённой процедуры в любом виде, появление нового сообщения в очереди сообщений, наступление определённого момента времени и т.д.

Система может реагировать как несколькими операциями (выполнять несколько функций) в ответ на одно событие, так и одной операцией в ответ на несколько событий разных типов.

Применение

Чёткое понимание операций системы (когда они выполняются и к каким эффектам приводят) является критически важным на всех этапах жизненного цикла разработки системы.

На этапе старта проекта по развитию существующей системы, эффекты помогают понять текущее наблюдаемое поведение, а сцепленность операций через ресурсы помогает спрогнозировать последствия планируемого изменения. Вместе это помогает предотвратить внесение регрессий, в виде нежелательных изменений в наблюдаемом поведении.

На этапе анализа, диаграмма эффектов помогает переформулировать требования в абстракциях будущей программы.

На этапе оценки это помогает понять что необходимо сделать (список операций) и насколько сложно это сделать (список событий, требуемых эффектов и целевых ресурсов).

На этапе проектирования операции и ресурсы становятся ключевыми блоками, правильная декомпозиция которых создаст основу для системы с низкой сцепленностью.

На этапе реализации сложность операций (определяемая количеством и типом обеспечивающих её ресурсов) и их зависимость через ресурсы помогают определить порядок выполнения работ и те работы, которые могут быть выполнены параллельно.

Нотация

Нотация с большой вероятностью претерпит косметические изменения в процессе бета-теста, а также в связи с привлечением профессионального дизайнера к полировке визуального языка перед релизом 1.0.

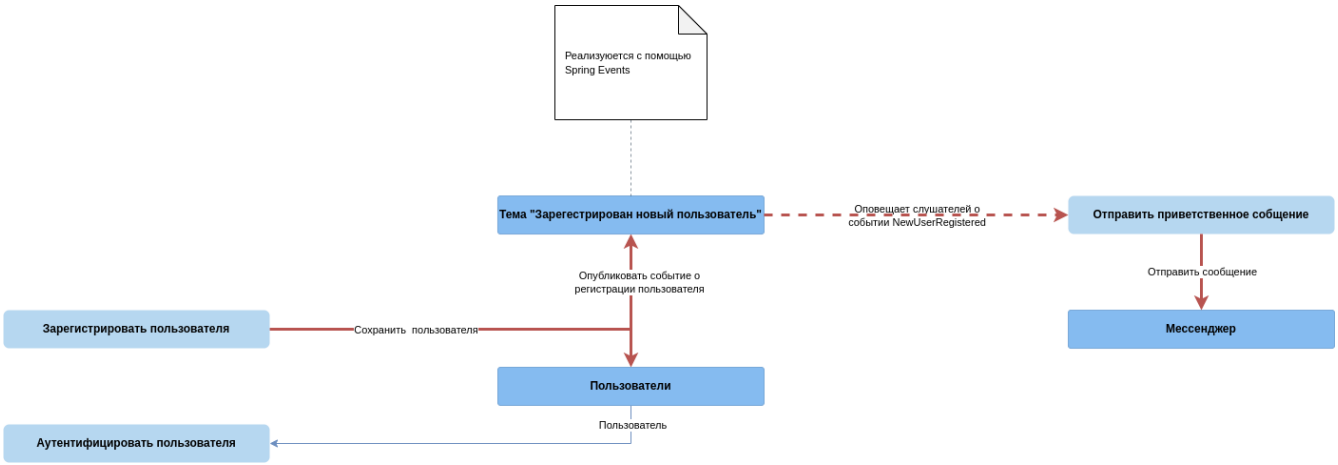
В основе визуального языка диаграммы эффектов лежит визуальный язык модели C4. С одной стороны, это позволяет встраивать диаграмму эффектов в модель C4 на четвёртом уровне - вместо кода. С другой наоборот, диаграмма эффектов может быть использована для декомпозиции системы на компоненты - третий уровень C4.

Диаграмма эффектов бывает двух типов - краткая и полная. Краткая содержит только обозначение эффектов и связанных ими операций и ресурсов. Полная нотация дополнительно включает события и их источники, внешние системы, обеспечивающие реализацию ресурсов и более полное описание всех элементов.

Нотации рассматриваются с помощью минимального примера визуализации функциональности регистрации и аутентификации пользователей в произвольной системе. После успешной регистрации пользователям отправляется приветственное письмо.

Краткая нотация

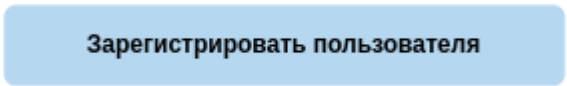
В краткой нотации диаграмма выглядит следующим образом (картинка кликабельна):



Теперь рассмотрим отдельные элементы

Операции

Операции обозначаются прямоугольником с именем операции:

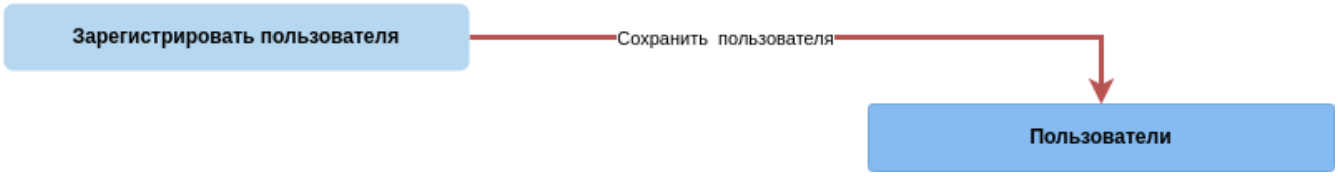


Ресурсы

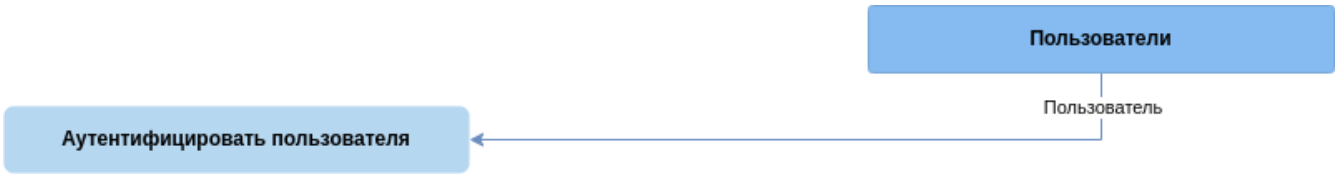
Ресурсы обозначаются прямоугольником с именем ресурса и цветом, отличным от цвета операции:

Эффекты

Эффект модификации ресурса обозначается "сильной" (более заметной) стрелкой от операции к ресурсу, с кратким описанием эффекта:



Эффект чтения ресурса обозначается стрелкой от ресурса к операции, с кратким описанием считываемых данных:



Эффекты вызова операций

Есть особый вид стрелок для эффектов вызова операций вследствие взаимодействия с ресурсами. Как правило, это ресурсы всевозможных шин событий и связанные с ними операции-обработчики. Такие связи отображаются прерывистой стрелкой того же стиля, что и стрелка эффекта записи с кратким описанием связи:

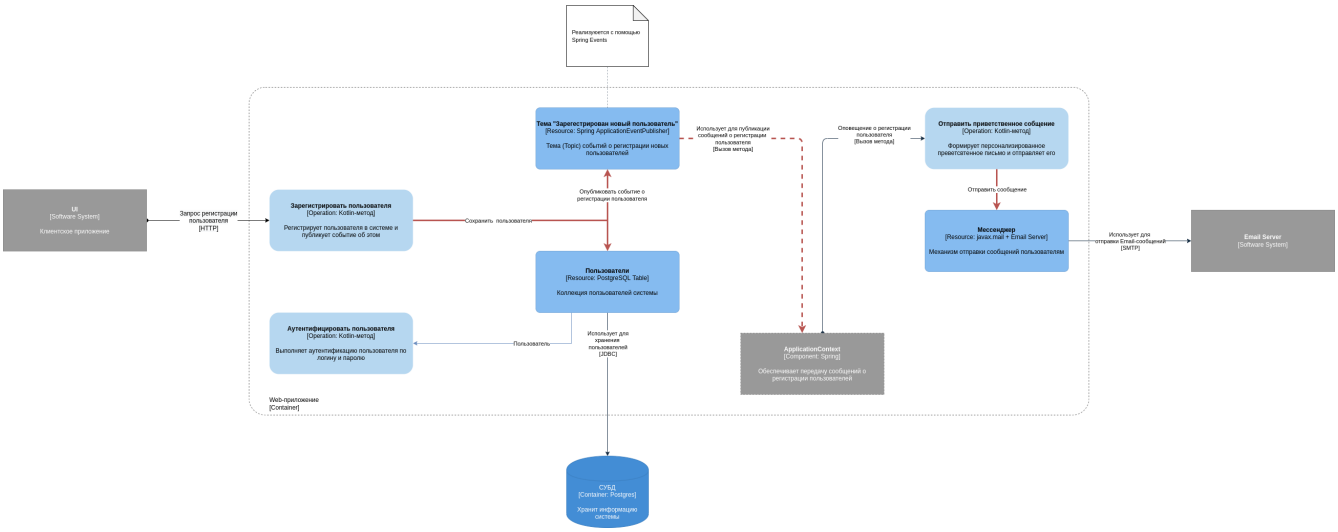
Примечания

Также на диаграмму можно помещать заметки и примечания, используя любую удобную нотацию. Я предпочитаю нотацию UML - "лист" с загнутым углом, связанный прерывистой линией с комментируемым элементом.

Это все элементы, составляющие ядро диаграммы эффектов.

Полная нотация

Теперь рассмотрим ту же функциональность, описанную в полной нотации:



В полной нотации появляются:

1. события
2. описание операций и ресурсов в формате модели C4
3. границы контейнера из C4. Обозначает границы процесса - всё, что находится внутри этих границ выполняется в памяти визуализируемого приложения
4. внешние системы, базы данных и компоненты из C4. Внешние системы могут быть как источником события, так и средством реализации ресурса

Расширять состав диаграммы можно постепенно, добавляя только те элементы, которые помогают в решении текущей задачи.

События

На мой взгляд, из дополнительных элементов наибольшую ценность имеют события. В полной нотации они обозначаются стрелкой от внешней системы к операции с кругом на

стартовом конце и описанием в формате C4. Но в промежуточной версии, внешнюю систему можно опустить и "подвесить" стрелку:

Описания

Затем блоки операций и ресурсов можно дополнить типом, способом реализации и описанием:

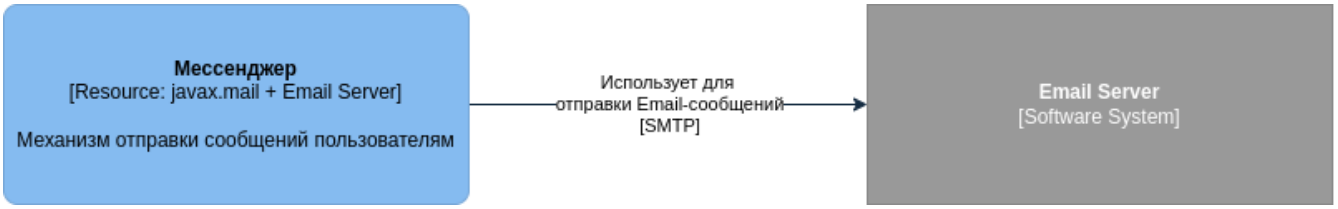
Внешние системы

Элементы, обозначающие границы системы и внешние системы полностью соответствуют нотации C4:

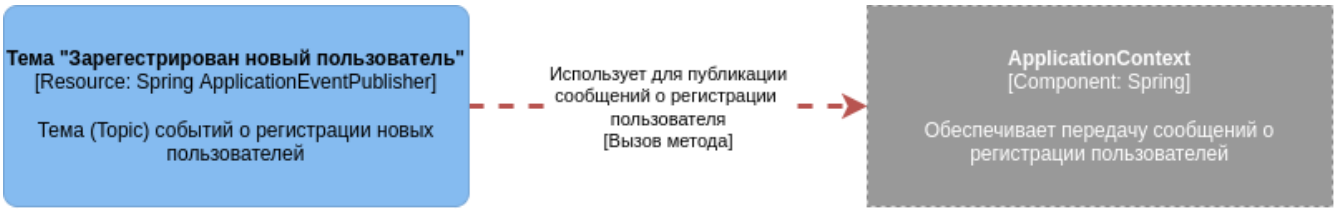
1. Границы системы отображаются прерывистым прямоугольником приглушённого цвета и подписью с именем контейнера
2. Управляемые внешние системы и базы данных обозначаются прямоугольником и символом "База Данных"
3. Неуправляемые внешние системы и компоненты обозначаются приглушёнными прямоугольниками
4. Неуправляемые базы данных обозначаются приглушённым символом "База Данных"

Внешние системы связываются с операциями посредством событий:

А ресурсы связываются с внешними системами посредством стрелок с описанием:



Ресурс может быть связан со сторонним компонентом, работающем в том же процессе:



Здесь приведена связь ресурса с эффектом вызова операции системы. В случае же если ресурс не обладает таким эффектом, то он соединяется со сторонним компонентом обычной стрелкой.

Выбор нотации зависит от решаемой задачи. Если надо быстро разбить систему на модули, или спланировать модификацию сложной или незнакомой операции - можно обойтись краткой нотацией. Если надо оценить проект для работы за фиксированную цену - лучше взять полную нотацию, чтобы минимизировать вероятность "потери" существенных деталей.

Ещё два критерия выбора нотации - срок жизни диаграммы и размер целевой аудитории диаграммы. Если планируете выкинуть диаграмму после анализа и никому не будете её показывать - можно обойтись краткой нотацией. Если же вы планируете возвращаться сами к диаграмме через длительный срок или публиковать её для ознакомления без вашего руководства - стоит как минимум добавить события и описания ресурсов и операций.

Я сам обычно начинаю с промежуточной нотации - краткой с событиями, и дополняю её по мере необходимости.

Приложение 1. Инструментарий

Благодаря базированию на визуальном языке модели C4, для построения диаграммы эффектов можно использовать любой инструмент с поддержкой C4.

Приложение 2. Философия эффекта

Эффект в программировании это всегда **действие** по изменению состояния какого-то транзистора. В эффекте чтения состояние транзистора памяти внешнего устройства через несколько промежуточных этапов попадает в регистр процессора (который является группой транзисторов).

В эффекте записи, наоборот, состояние регистра процессора переносится в транзистор внешнего устройства. На этом эффект может быть завершён, в случае жёсткого диска, либо это состояние может быть "аналогизировано" - превращено в физический процесс (например, световую волну или движение ноги робота). Так эффект работы программы становится наблюдаемым поведением.

Вообще "неэффектов" не существует - любое действие в программе выражается в изменении состояния транзисторов. В фон Нейманавской архитектуре, по крайней мере.

При этом существует функциональная парадигма программирования, которая характеризуется акцентом на функциях без эффектов. Но в современном компьютере даже идеально чистая вызывающая функция записывает данные в память, выделенную для стэка, и ожидает, что идеально чистая вызываемая функция их считает, а потом запишет свой результат.

Я ни разу не встречал формального определения того, что считается эффектом, а что "неэффектом". Но, по-видимому, общепринятое мнение таково, что изменение регистра процессора и стэка программы эффектами не считается, а любые изменения начиная с кучи программы и далее - считаются. То есть разница между эффектом и "неэффектом" в области видимости. Или, другими словами, в количестве наблюдателей поведения.

Высокоуровневый эффект может проходить через несколько этапов переноса, посредством чтения и записи. Например, эффект "Отправить пуш уведомление" пройдёт такой путь: сначала информация переносится через кэши из процессора в память программы, потом в память ОС, потом в память сетевой карты, потом через память нескольких роутеров и серверов в память сетевой карты другого компьютера (смартфона), там обратно в память программы, а оттуда, опять же через несколько слоёв, в память экрана, где состояние транзистора "аналогизиурется" в свечение пикселя. И где-то попутно этот эффект заодно осядет на транзисторах диска БД пуш-сервиса.

Приложение 3. Реализация концептуальной модели в коде

Все элементы, описанные в концептуальной модели, транслируются непосредственно в код: события и операции - в методы, ресурсы - в классы, эффекты - в вызовы методов.

Операции всегда транслируются в методы классов слоя сервисов приложения - методы, определяющие публичный интерфейс модуля. При реализации этих методов желательно сохранить очевидность эффектов выполнения операции, присущую диаграмме.

Ресурсы превращаются в структуру данных и коллекцию методов работы с ней - классы Spring Data агрегата и репозитория, классы события и ApplicationEventPublisher-а (или обёртки вокруг него), классы REST API модели и клиента и т.п. В контексте бэкэндов информационных систем, самыми распространёнными видами ресурсов являются:

1. любые постоянные коллекции данных - таблицы в реляционной СУБД, коллекции в документной СУБД и т.д.
2. REST API внешних сервисов
3. любые очереди сообщений/шины событий
4. изменяемые структуры данных, доступные через глобальные переменные

События превращаются в методы, передаваемые фреймворку для последующего вызова - метод Spring-ового RestController-а, Swing-овый EventListener, реализация Runnable для таймера и т.д. Если говорить о бакэндах информационных систем, то самыми распространёнными видами событий являются:

1. Получение запроса по сети (@RestController + @*Mapping в случае разработки на Spring). Сейчас популярностью пользуется протокол запросов в REST-стиле, но SOAP, gRPC, CORBA и т.п. так же попадают в эту категорию.
2. Появление сообщения в очереди (@JmsListener).
3. Доменное событие или событие приложения (@EventListener)
4. Наступление определённого момента времени (@Scheduled). Два основных типа таких событий:
    a. наступление заранее известного момента времени (например, полуночи вторника)
    b. истечение определённого времени с момента в прошлом (например, истечение суток с момента создания предыдущего бэкапа).