Kotlin DSL для описания сценариев автоматизации бизнес-процессов в телефонии и движок для их исполнения

Креславский Кирилл Дмитриевич

Научный руководитель: Кузькин Виталий Андреевич, доктор физико-математических наук, профессор, департамент информатики

Научный консультант: Иванова Марина Геннадьевна, ведущий инженер, АО Nexign

Санкт-Петербургская школа физико-математических и компьютерных наук

НИУ ВШЭ – Санкт-Петербург

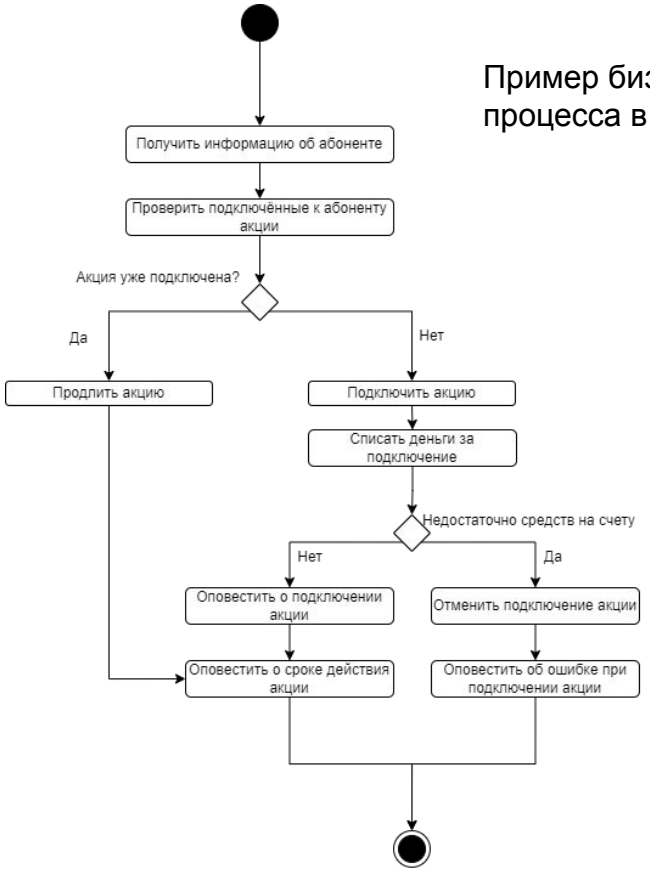
Введение в область

DSL – Domain Specific Language, то есть в нашем случае буквально язык программирования внутри языка программирования, предоставляющий более верхнеуроневую абстракцию.

Хорошие примеры Kotlin DSL есть тут github.com/zsmb13/VillageDSL

```
val v = village {
    house {
        +Person("Emily", 31)
        +Person("Hannah", 27)
        +Person("Alex", 21)
        +Person("Daniel", 17)
    }
}
```

```
val v = village containing houses {
    house with people {
        "Emily" age 31
        "Hannah" age 27
        "Alex" age 21
        "Daniel" age 17
    }
}
```



Пример бизнес процесса в телефонии

Для кого я это делал

Nexign — российская компания-разработчик OSS/BSS-систем и решений для цифровизации бизнеса, первый в России разработчик биллинга для телекоммуникационных компаний

Клиентами Nexign являются более 50 телеком-операторов из 16 стран, в том числе «Мегафон», Moldcell, МТС, «Ростелеком», Kcell и Turkcell

Ранее известна как “Петер-Сервис”

Цели и задачи

Глобальная цель: разработать эффективный Kotlin DSL для лаконичного, читаемого написания сценариев и небольшой движок для их исполнения, за счёт чего значительно снизить число ошибок, возникающих по причине "человеческого фактора" (опечаток, несовместимости типов и т.п.)

- Изучить существующие аналоги, сделать анализ их преимуществ и недостатков
- Разработка DSL с постепенным добавлением нового функционала
- Разработка движка для исполнения в Production и Testing режимах
- Покрытие автотестами
- Проверка концепции, функциональности. Оценка удобства использования заинтересованными пользователями

Сравнение с аналогами 1

Фреймворки / библиотеки и продукты для разработки и описания бизнес-процессов / сценариев

Cadence – движок для BPMN процессов на **Go** с библиотеками на Go, **Java**, Python и Ruby. **Open source**. Огромный, покрывает весь BPMN, отчего очень сложный интерфейс. Очень много настроек всего и вся. Сверхрасширяемый под всё что угодно. Одним словом – Overkill	Temporal – фреймворк для BPMN, основанный на Cadence, более простой в эксплуатации и настройке, тоже **Open source,** тоже **Java**,	iWF – фреймворк для BPMN для Cadence, более простой, чем у самого Cadence и более простой, чем Temporal.

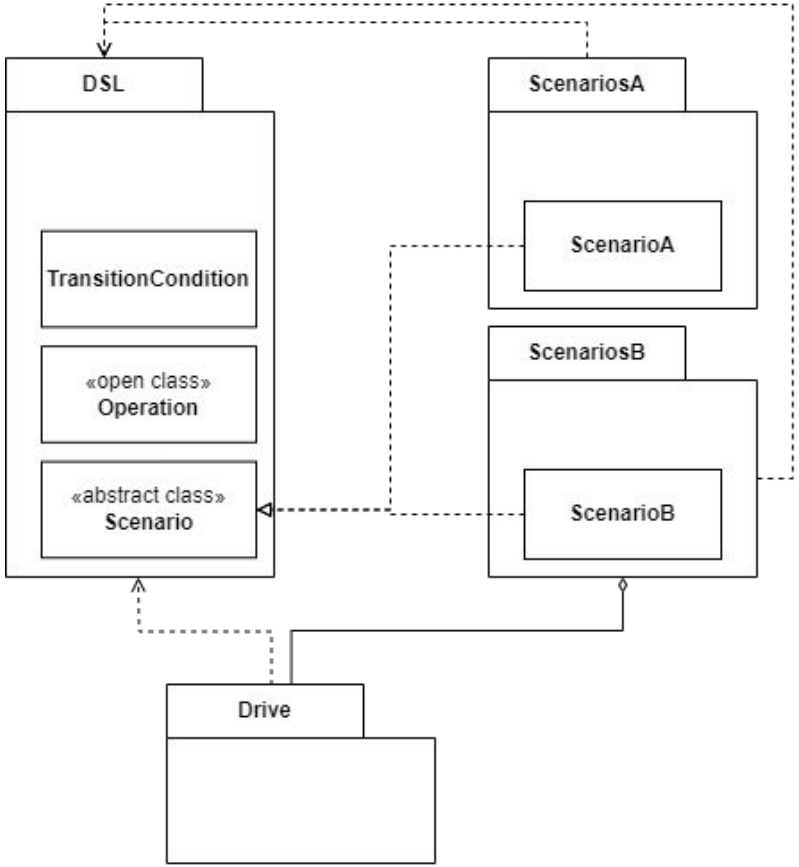
Общая проблема – размазанное по разным местам определение переходов, нет единого места описания сценария, не человеко-читаемое

Сравнение с аналогами 2

Фреймворки / библиотеки и продукты для разработки и описания конечных автоматов

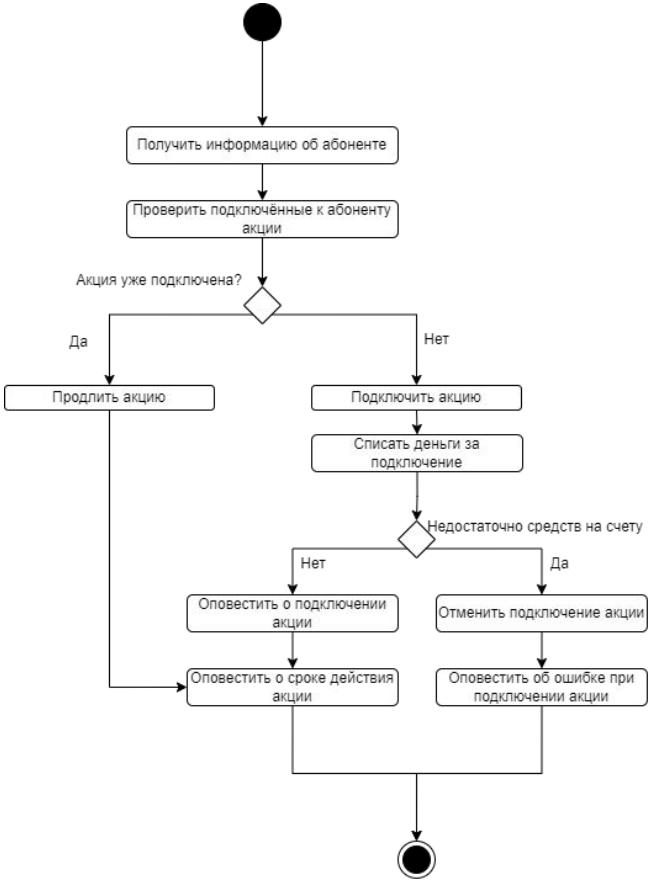
Название	Краткое описание	Наличие движка для исполнения	Автогенерация картинки с описанием по коду
KStateMachine	Более компактный фреймворк, но всю работу операций и все переходы нужно описывать в одном месте	+/-	-
StateMachine (tinder.com)	Чистый State Machine Builder, где всю полезную работу можно положить только в side effect'ы переходов, сами состояния это просто состояния. Тоже получается огромное полотнище.	-	-
KFSM	Неплохой фреймворк, предоставляет как раз Kotlin DSL, но описание даже маленькой State Machine занимает бесконечно много места.	-	+

Наше решение



Пакетам сценариев требуется знать только то, как эти сценарии описывать, то есть DSL и совсем не требуется знать, как устроены кишки движка. Движку требуется знать только то, как эти сценарии понимать, то есть DSL. А сами сценарии он может кушать в виде уже готовых JAR.

Наше решение



Kirill Kreslavski *

```
class ExampleScenario(store: MutableMap<String, Any>) : Scenario(store)  {

    override val specification: Specification = specification (
            operation: getAbonentInfo next checkAbonentActions binary { this: BinaryChoice
                yes( op: prolongAction next notifyAboutActionTimePeriod next end)
                no( op: activateAction next writeOffMoney multiple { this: MultipleChoiceBuilder
                    +(YES to (cancelActionActivation next NotifyAction("error when activating action") next end))
                    +(NO to (NotifyAction("action activation") next notifyAboutActionTimePeriod))
                })
            }
    )
```

Kirill Kreslavski *

```
    companion object {
        val getAbonentInfo = GetAbonentInfo()
        val checkAbonentActions = CheckAbonentActions()
        val prolongAction = ProlongAction()
        val activateAction = ActivateAction()
        val writeOffMoney = WriteOffMoney()
        val cancelActionActivation = CancelActionActivation()

        val notifyAboutActionTimePeriod = NotifyAction("action time period")
        val end = object : Operation() {
            override val func: Scenario.() -> TransitionCondition = { STOP_EXECUTION }
        }
    }
}
```

Решённые задачи

- Разработан базовый вид DSL
  - Создана базовая архитектура
  - Лаконичное описание операций
  - Лаконичное описание переходов
- Разделены сущности DSL, движка и пакета сценариев
  - Движок собирается отдельно и использует DSL как библиотечную зависимость
  - DSL ничего не знает о внутренней реализации движка
  - Для написания сценариев нужны только средства DSL
- Автоматическое построение человеко-читаемого описания (текста на английском) сценария по его спецификации
- Сохранение человеко-читаемого описания прохода исполнения сценария

В процессе

- Доработка DSL
  - Упрощение описания обработки верхнеуровневых ошибок сценария
- Доработка движка
  - Перенос функциональности рутинга и обработки ошибок в движок, а также их рефакторинг
  - Добавить новый механизм обработки верхнеуровневых ошибок сценария
  - Добавить возможность предоставления внешних зависимостей через Kotlin Context (experimental)
  - Добавить возможность автоматической генерации xml спецификации сценариев и изображений по ним
  - Аналогичное для прохода исполнения сценария
- Написание тестов для всех пакетов
- Написание документации

Результаты

репозиторий