DSL-конструкции языка Kotlin и архитектура как код

Николай Поташников

Руководитель проектов, IT-архитектор
КУРС-ИТ

consulting@yandex.ru @nmpotashnikov

JPoint 2023

Содержание

- Описание подхода, примеры
- Детали реализации

Решаемые задачи

- Новый человек в команде
- … хуже — у заказчика
- … совсем плохо — по информационной безопасности

Типовые ожидания от архитектурного описания информационного продукта (системы)?

- Каждое лицо должно получать необходимую для него информацию, но согласованную с общей моделью
- Должна обеспечиваться трассируемость архитектурных решений: от стрелочки на диаграмме до реализации в коде и ответственных (безответственных)
- Возможность автоматического тестирования архитектурных решений
- Должна обеспечиваться удобная работа с историей изменений (как мы привыкли работать с коммитами в коде)

Что мы используем для управления архитектурой

- Язык описания модели — Kotlin
- IDE — IntelliJ IDEA
- Язык простой текстовой разметки — Asciidoc[tor]
- Система контроля версий/CI — Github/Github Actions

Пример 1. Диаграмма контекста C4

C4 модель — простой метод графической записи (нотации) для моделирования архитектуры программных систем

Уровни диаграмм:

- **C**ontext
- **C**ontainers
- **C**omponents
- **C**ode

Strukturizr DSL (https://github.com/structurizr/dsl) — язык описания архитектуры на основе подхода Диаграммы как код

Пример 2. Технология подключения пользователя к ресурсам системы

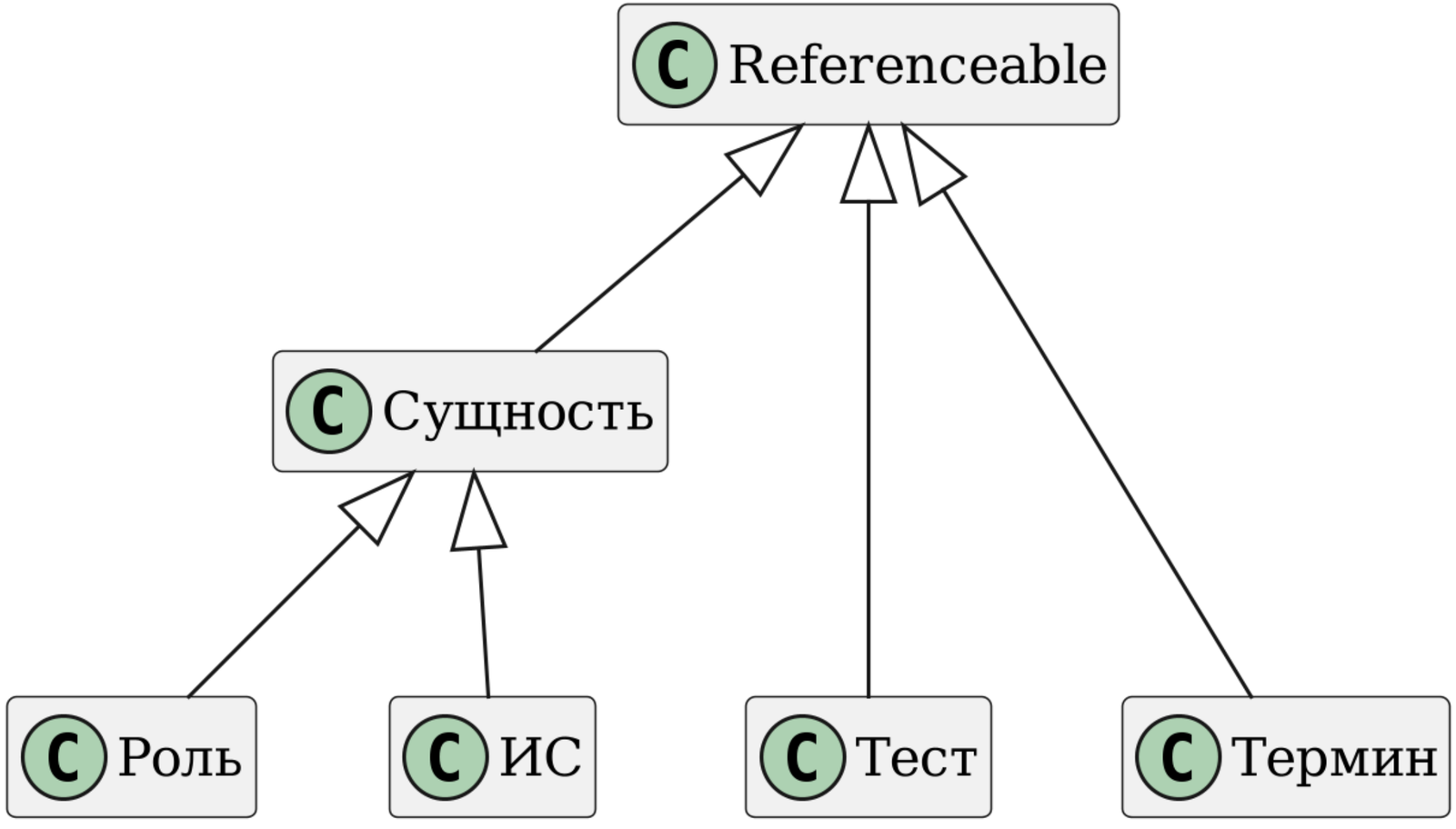
Текстовое описание алгоритма, в соответствии с которым пользователь получает доступ к ресурсам информационной системы

Demo

Эволюция архитектурного описания

Проектирование	Разработка	Сопровождение
Документация полностью сделана вручную	Элементы документации создаются автоматически	Вручную создаются только отдельные элементы документации

Базовые подходы к описанию, примерная диаграмма классов



Описание объекта

```
object ИСы : Dictionary() {
    object НСИ {
        val нси = ИС {
            система краткоНазывается "ИС НСИ"
            система называется "Информационная система обработки нормативно-справочной информации"
        }
        val ип1 = ИнтеграционныйПоток {
            поток определяетОтправителя Система.система определяетПолучателя нси
            поток краткоНазывается "Запрос новых сведений о контрагентах"
        }
        val ип2 = ИнтеграционныйПоток {...}
        val сц1 = ИнтеграционныйСценарий {
            сценарий краткоНазывается "Получение сведений о контрагентах"
            +ип1
            +ип2
        }
    }

    object СЭД {...}
```

Настройка DSL

Определение сущности

```
class ПрограммныйКомпонент(иниц: ПрограммныйКомпонент.() -> Unit) : Сущность() {
    val компонент = this
    init {
        apply(иниц)
    }
}
```

Определение списков

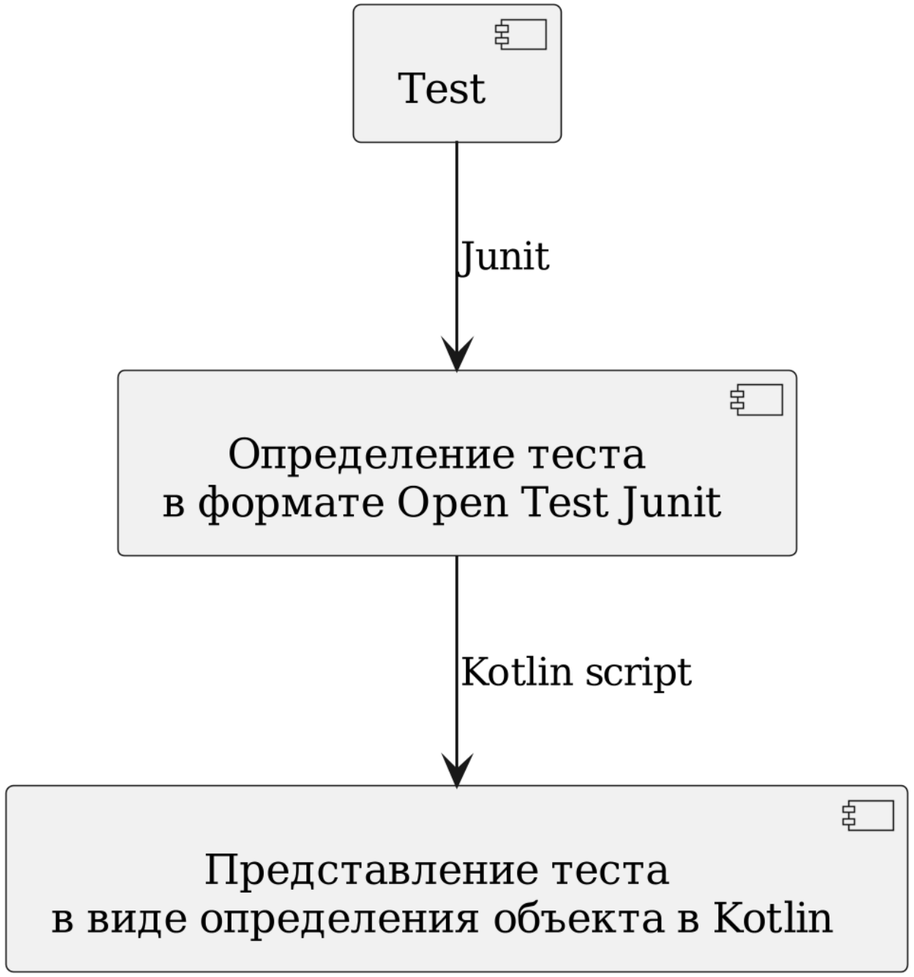
```
open class SubElements<T> : ArrayList<T>() {
    operator fun T.unaryPlus() {
        this@SubElements.add(this)
    }
}
```

Настройка DSL (продолжение)

Определение атрибута сущности

```
infix fun определяетОтправителя(отправитель: ИС): ИнтеграционныйПоток {
    this.отправитель = отправитель
    return this
}
```

Включение в модель тестов — общий алгоритм



Пример создания описания тестов — исходные данные

Спецификация теста

```fun
@Test
@DisplayName("Тестирование подключения к LDAP")
fun someLdapTest() {
    // Body
}
```

Отражение теста в формате Open test reporting

```
<e:started id="12" name="Тестирование подключения к LDAP" parentId="9" time="...">
  <metadata>
    <junit:uniqueId>
      [engine:junit-jupiter]/[class:ru.curs.kbmmock.ArchDemoTest]/[method:someLdapTest(
    </junit:uniqueId>
    <junit:type>TEST</junit:type>
  </metadata>
  <sources>
    <java:methodSource className="ru.curs.kbmmock.ArchDemoTest" methodName="someLdapTes
  </sources>
</e:started>
```

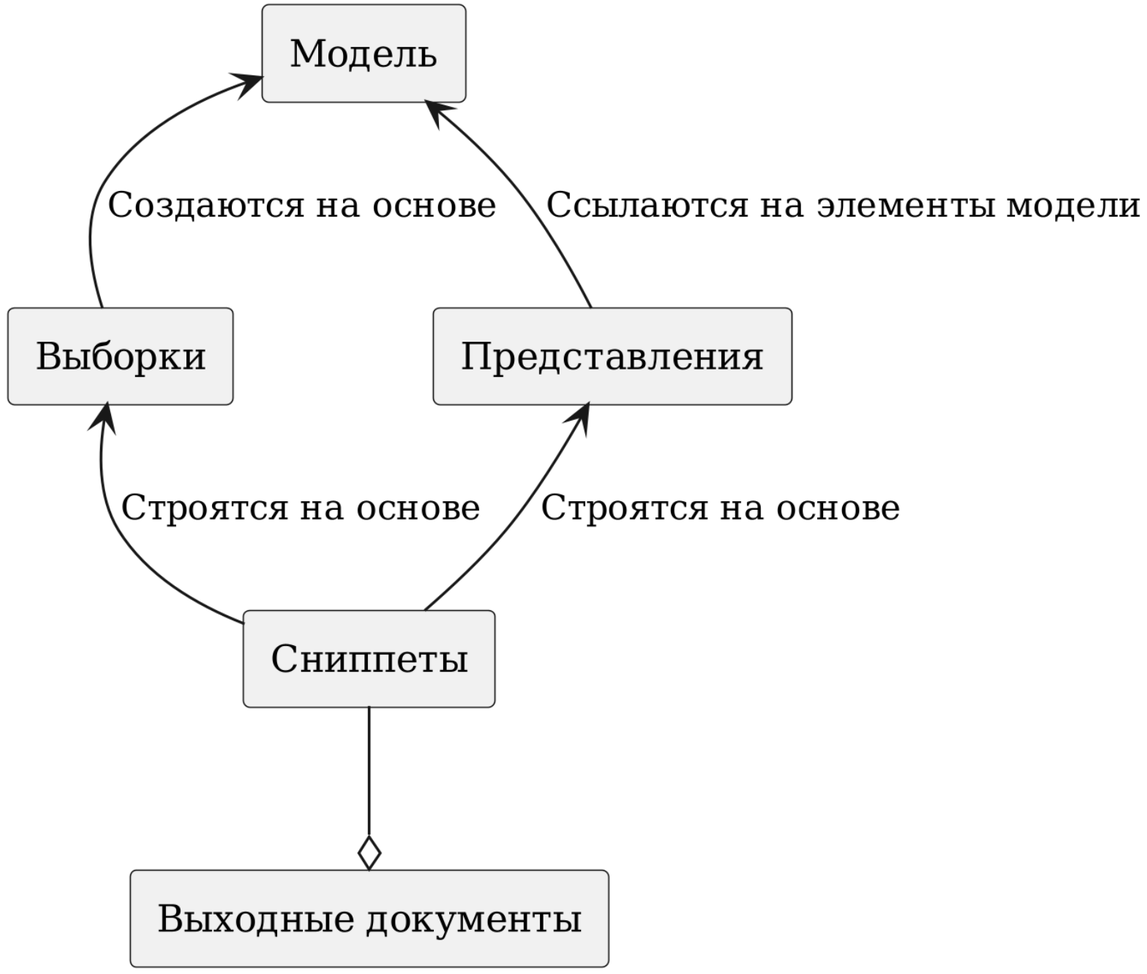
Продолжение примера — описание теста для включения в описание архитектуры

```
val someNewLogoutTest = Test {
  nom = "Тестирование выхода пользователя"
  testJetBrainsURL = "jetbrains://idea/navigate/reference?project=kotlin-business
   -modelling-demo&fqname=ru.curs.kbmmock.ArchDemoTest.someNewLogoutTest"
}
```

Пример — код создания описания теста (Kotlin script)

```
testDictionary.apply {
    events.started MutableList<Started>
        .filter { it.metadata != null && it.metadata.type == "TEST" } List<Started>
        .filter {
            it.sources?.methodSource != null && it.sources.methodSource.className ==
                    "ru.curs.kbmmock.ArchDemoTest" }
        .forEach {
            +TestRecord(
                    className = it.sources!!.methodSource!!.className,
                    methodName = it.sources.methodSource!!.methodName,
                    id = it.sources.methodSource.methodName,
                    name = it.name,
                    jetBrainsURL = "jetbrains://idea/navigate/reference?project=" +
                            "kotlin-business-modelling-demo&" +
                            "fqname=${it.sources.methodSource.className}" +
                            ".${it.sources.methodSource.methodName}"
            )
        }
}
```

Получение выходных форматов — общая схема



Генерация выборок

Пример получения списка терминов

```
fun fullTermList(arch: Архитектура): List<TermView> {
    return (listOf<TermView>() +
            arch.термины.map {
                TermView(Utils.getRefAttributeAsAnchor(it), it.определение!!, it.nom)
            } +
            arch.сопряженныеИС.map {
                TermView(Utils.getRefAttributeAsAnchor(it), it.наименование!!, it.nom)
            } +
            TermView(
                Utils.getRefAttribute(arch), arch.наименование!!, arch.nom
            )
            ).sortedBy { it.sortBy }
}
```

Использование типобезопасных конструкторов для создания сниппетов

Преобразования списка терминов в формат Asciidoc

```
table {
    attr("cols", "1,2")
    fullTermList(arch).forEach {
        td { p { +Attribute(it.nameAsAttribute) } }
        td { p { +it.description } }
    }
}
```

Примерный вид выходного Asciidoc-документа

```
[cols="1,2"]
|===
|термин
|определение
...
|===
```

Использование атрибутов Asciidoc для обозначения сущностей

Вид атрибутов, создаваемых из модели, в сниппете

```
:is-nsi-nsi-nom: <<IS-NSI-NSI, ИС НСИ>>
```

Использование атрибутов в тексте

```
Краткое наименование -- {is-nsi-nsi-nom}
```

Результат

Краткое наименование — ИС НСИ

Пример представления

Диаграмма контекста C4

```
val contextC4 = C4Model().apply {
    +C4ModelRel(Роли.роль1, демоАрхитектура, "использует")
    +C4ModelRel(Роли.роль2, демоАрхитектура, "использует")
    +C4ModelRel(демоАрхитектура, ИСы.НСИ.нси, "получает контрагентов", ИСы.НСИ.сц1)
    +C4ModelRel(демоАрхитектура, ИСы.СЭД.сэд, "получает сведения о документе", ИСы.СЭД.
}
```

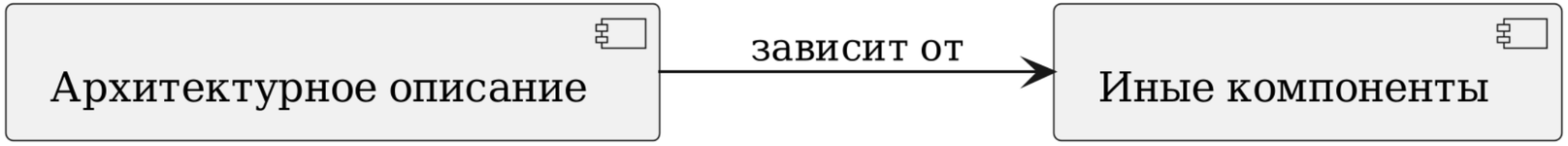
Использование идентификаторов

Переменные	lowerCamel	`val администраторСистемы = …`
Идентификаторы	UPPER-KEBAB	`РОЛЬ-АДМИНИСТРАТОР-СИСТЕМЫ`
Атрибуты Asciidoc	lower-kebab	`{роль-администратор-системы-nom}`

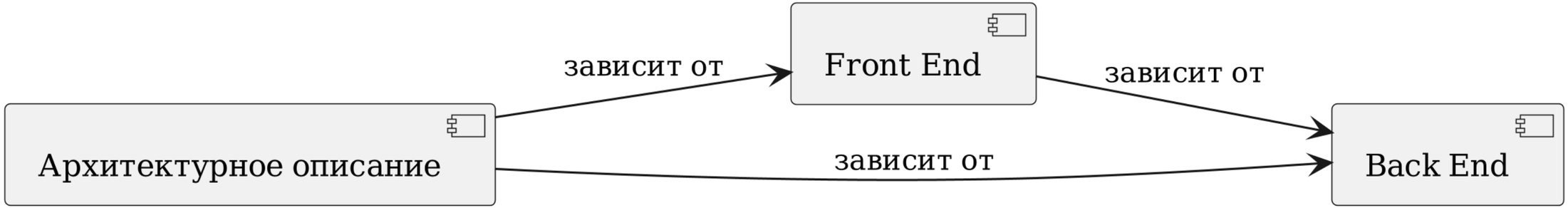
Для перекодировки используется транслитирация ISO 9:1995, ГОСТ 7.79-2000 Система Б со следующими исключениями

- ц — cz (всегда) вместо c, cz
- ь — q вместо `
- ы — yh вместо y`
- ъ — qq вместо ``
- э — eh вместо `
- х — x и это не исключение

Архитектурное описание как программный компонент



Пример типового проекта с Back End и Front End



Выводы

- Архитектурное описание как код можно рассматривать как программный компонент (модуль) информационного продукта (системы)
- Подход «Архитектура как код» обеспечивает постоянную синхронизацию состояния информационного продукта (системы) и его архитектурного описания, эффективен на этапе рабочего проектирования (кодирования) и сопровождения
- Kotlin содержит необходмые средства для создания архитектурных описаний: поддерживает наглядные DSL-конструкции, обеспечивает статическую типизацию, требует минимум шаблонного (boilerplate) кода. IDE IntelliJ Idea обеспечивает удобную среду работы с архитектурным описанием