Архитектурное программирование Языки Арс и Арвиль

Алексей Недоря, июль 2024

Благодарности: А. Канатов, Д. Соломенников, Е. Зуев, И. Беспальчук

Архитектурное программирование (АРП)

АРП - это **технология программирования**, обеспечивающая **корректность описания архитектуры** программной системы (ПС) в течении всего времени жизни ПС.

Архитектура определяет:

1. Составные **части** ПС.
2. **Структуру** или Устройство - как расположены/вложены части.
3. **Взаимодействие** частей.

В АРП **описание архитектуры** является **входными данными** для компиляторов и/или систем сборки.

- описание архитектуры
- исходный или бинарный код
- ресурсы

Изменение описания приводит к изменению исполняемой ПС (ИПС), а изменение архитектуры ИПС может быть сделано только через изменения описания.

Не должно быть зависимостей и взаимодействий, кроме описанных в архитектуре.

Область использования АРП

Долгоживущие, развивающиеся, большие программные системы:

- упрощение системы и упрощение развития
- расширяемость
- управляемость
- удобство обслуживания (maintainability)
- стоимость разработки и стоимость обслуживания

Попытки двигаться в сторону АРП были, например, на базе UML, но, ни одна из известных мне, не пыталась сделать прямую и неразрывную связь между описанием архитектуры и исполняемой ПС.

A2 - реализация АРП

Программная система состоит из **схемы** и набора **компонент**. Компонента называется **Арка** от **ар**хитектурная **к**омпонент**а.**

Схема описывает

- **структуру**,
- **настройку**
- и **взаимодействие** арок

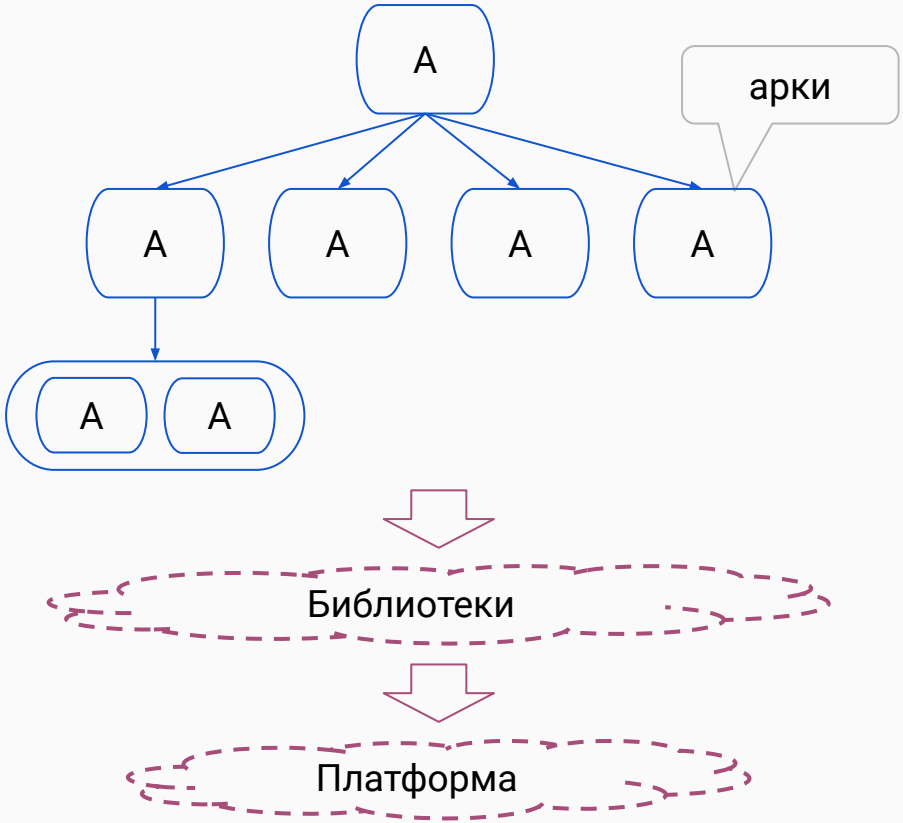
Арка (определяет действия):

- может быть **простой** (написанной на языке программирования)
- или **сборной** (то есть состоящей из схемы и набора других арок).

Интересные возможности:

- плавный переход от динамики к статике (gradual static)
- частичная оптимизация (де-аркаизация)
- прототипирование, горячая замена (hot reload)

A2: Структура программной системы



Программная система:

- Структура из **Арок** (дерево или ациклический граф?)
- Арки используют **Библиотеки** (импорт)
- Библиотеки используют **Платформу** (переносимый слой)

Взаимодействие арок на этом рисунке не отражено, только структурные связи.

A2 и языки программирования

Для A2 нужно уметь

- Разрабатывать схемы
- Разрабатывать простые (программные) арки, собирать их них высокоуровневые арки.

Можно ли использовать существующие языки (нотации)?

- Для схем: **Нет**. Задача новая, надо делать новую нотацию.
- Для программных арок: **Можно**. Но сначала лучше сделать простой, но точно нацеленный, специализированный язык. Это поможет найти основные конструкции, выработать терминологию и реализовать прототипы.

- **Арс** - язык описания **Ар**хитектурных **С**хем (преимущественно декларативный)
- **Арвиль** - язык разработки арок (**Ар**хитектурный Три**виль**)

Пример 1: Привет, мир

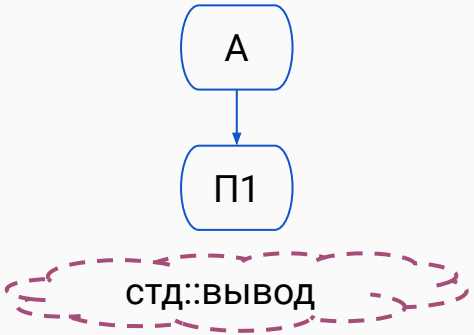
```
арс :: (
    арка (
        конст Адрес = "арки::примеры/пример1"
    )
)
```

```
арка пример1

импорт "стд::вывод"

вход (_) {
    вывод.ф("привет, мир\n")
}
```

```
Привет, мир
```



- Программа состоит из двух арок:
  - арс#1: одна стандартная (папка, контейнер) - аркада
  - арс#2..4: арка из “арки:примеры/пример1”
- Арка пример1 импортирует библиотеку: стд::вывод
- *Схема (в каком-то смысле) это makefile*

Пример 2: Привет, коллеги

```
1  арс :: (
2      арка мир (
3          конст Адрес = "арки::примеры/пример2"
4          контакт Имя = "коллеги"
5          протокол { фн привет()  }
6      )
7      вход мир.привет()
8  )
9
```

```
1   арка пример2
2
3   импорт "стд::вывод"
4
5   контакт {
6       Имя: запрос (): мб Строка
7   }
8
9   фн (а) привет*() {
10      вывод.ф("Привет, $;!\n", а.Имя())
11  }
```

```
Привет, коллеги!
```

- арс#7: вызов метода арки при запуске программы (этой арки)
- арс#5: протокол, который должен быть реализован аркой (утиная типизация, динамика, но может быть статика)
- арс#4 - реализация контакта, описанного в арвиль#6

Контракт: эволюция

Процедурные языки

```
1  extern int puts (const char *__s);
2
3  int main() {
4    puts("hello\n");
5  }
```

Контракт на функцию:

1. Есть функция, с нужной сигнатурой

Модульные языки

```
1  модуль x
2  импорт "стд::вывод"
3  вход {
4    вывод.ф("привет\n")
5  }
```

Контракт на модуль:

1. есть импортируемые модули
2. есть экспорт с нужными сигнатурами

Арс/Арвиль

Контракт на арку:

1. есть импортируемые модули
2. есть экспорт (арка реализует нужные протоколы)
3. контакты подключены

Пример 3: Контакты и гибкость

```
арс :: (
    :: ввод-вывод (
        арка вывод (
            конст Адрес = "арки::примеры/стд-вывод"
            доступ напечатать строку (с: Строка) через Вывод@".."
        )
        арка имена (
            конст Адрес = "арки::примеры/имена"
            доступ дай имя(): мб Строка через Имя@".."
        )
    )
    арка мир (
        конст Адрес = "арки::примеры/пример4"
        контакт Имя: запрос (): мб Строка через Имя@"../ввод-вывод"
        контакт Сообщить: уведомление (с: Строка) через Вывод@"../ввод-вывод"
        протокол {
            фн привет()
        }
    )
    вход мир.привет()
)
```

```
арка стд-вывод

импорт "стд::вывод"

фн (а) напечатать строку*(с: Строка) {
    вывод.ф("$;\n", с)
}
```

```
арка имена

тип Строки = []Строка

пусть {
    список = Строки["Вася", "Петя", "Коля"]
    № := 0
}

фн (и) дай имя*(): Строка {
    пусть тек = и.список[и.№]
    и.№ := (и.№ + 1) % длина(и.список)
    вернуть тек
}
```

```
арка пример4

импорт "стд::строки"

контакт {
    Имя: запрос (): мб Строка
    Сообщить: уведомление (имя: Строка)
}

фн (а) привет*() {
    а.Сообщить(строки.ф("Привет, $;!\n", а.Имя()))
}
```

Что дает Архитектурное программирования

1-й уровень:

- Сохранение архитектуры в течении всей жизни программной системы.

2-й уровень:

- Новый уровень абстракции - **программа отделена от кода**. Архитектор **делает программу** на уровне требований, частей и их взаимодействия
- Кардинальное снижение сложности - архитектор работает не с миллионом строк кода, а с описанием, которое в тысячи раз меньше. Программист тоже работает с не с программой, а с компонентой, тоже в тысячи раз меньше..

3-й уровень:

- точки роста, замена компонент, плавное развитие программных систем
- переносимость

Заключение. Ссылки

Что дальше

- Арс: язык и компилятор
- Графические компоненты
- Большой пример (IDE?)
- От прототипа к системе

Для исследователей: новая точка зрения на

- описание архитектуры и компоненты
- анализ и оптимизацию
- верификацию
- ?

Публичный репозиторий	https://gitflic.ru/project/alekseinedoria/trivil-0
Арвиль: описание языка	https://gitflic.ru/project/alekseinedoria/trivil-0/blob?file=doc%2Fарвиль%2Fописание%2Fарвиль.pdf
Арс: описание языка	https://gitflic.ru/project/alekseinedoria/trivil-0/blob?file=doc%2Fарс%2Fописание%2Fарс.pdf
Ворчалки о программировании	http://алексейнедоря.рф/

Вопросы и ответы

?