Tuesday, April 26, 2022 11:08 AM

Лекция 12

Продолжение лекции 11

Интерпретатор

Декомпозиция: пр-ма умеет разбирать и интерпретировать, менять внутр. состояние (сит)

Компоненты: инт-р команд, API, состояние.

Коннекторы: прямой доступ к памяти, доступ по ссылке, вызов процедур

Элементы данных: команды

Топология: трехуровневая (слойная) архитектура

Ограничения: нет

Обеспечивает: гибкое поведение с дин. набором команд

Типичное применение: сис-мы проф-ния, настройки оборудования

Описывает: инт-ции домен. исх. кода

Примеры: сис-мы макросов (Excel) LISP
много разных интерпретаторов

+ Топологически равен экспертной системе.

Мобильный код

Декомпозиция: код/состояние передается

Компоненты: инт-р/комп-р, компон. отправки, комп-т развертывания

Компоненты: код, ...

Обеспечивает: дин адаптивность системы

Типичные применения: обработка больших объемов данных у разных удаленных регионов

Опасности: безопасность (уязвимо для злона)

Примеры: Java Script, Active X, VB Script, grid-вычисления, Java Applets, Adobe Flash

Суть: код передаётся, а затем уже обрабатывается (локально)

Неявный вызов

- взаимодействуют благодаря „событиям"
- комп.-слушатели регистрируются

Коннекторы: явная и неявная реакция на событие

- комп., которые объявляют события, не беспокоятся о том, что дальше + не знают ничего о слушателях

+ комп СЛАБО связаны
+ многократно исп-ть компоненты
+ легко развивать систему
+ сис-ма может обновляться во время исп-ния

- сложная арх-ра
- комп. не полностью контр-т систему
- неизвестно, какие именно комп. ответят, как и когда
- запутанное протекание процессов
- неизвестно, как система поведет себя в развернутом состоянии в конкретной ситуации

Фильтры-и-Транзакторы

Декомп. подписчики подписываются
трансл. держат список ч.
синх. и асинхронно.
Сообщения могут содержать данные

Компоненты: издатель подп и трансл.,
прокси-комп.

Коннекторы: вызов процедур,
сетевые протоколы,
событийные сис-мы

Типичное применение: рассылка данных, GUI

Опасности: проблемы эффективности,
сложная арх-ра

Примеры: реактивные каркасы

Событийная архитектура
Event-based

Декомп: есть общая шина событий –
отдельный компонент

Компоненты: те же ↑

Коннекторы: шина событий

Эл-ты данных: событийная инф-ия

Топология: шина + компоненты

Ограничения: нет

Обеспечивает: масштаб-ть и расшир-ть
арх-ры, для гетерогенных
приложений

Типичное применение: GUI, ...

Пиринговая арх-ра
Peer-to-peer

Декомп. много нед-х компонентов,
но децентрализ важно

Компоненты: независимые пиры

Коннекторы: сетевые протоколы

Концентры.

Типичное применение: система обмена данных

Примеры: Bit Torrent-трекеры, eDonkey, Bitcoin, Ether, I2P, Tor, Microsoft delivery optimization, ...

* компоненты одинаковые

Опасности: медленная работа, не безопасная, но отказоустойчивая арх-ра,

Гетерогенные архитектуры

Примеры: RESTful, CORBA, стиль Се

* сочетают в себе черты разных архитектурных стилей

Сервис-ориентированные архитектуры

Декомпозиция: приложение „собирают" из многочисленных сервисов, которые взаимодействуют между собой

SOA-приложения не обязательно сетевые и распр.

+ комп. слабо связаны
+ можно разраб независимо
+ исп-ть готовые сервисы
+ устойчивость
- накл. расходы на взаим-вие
- арх-ра сложно
- много модификаций при слабой согл. команд
- проблемы надежности и конф-ти

Функциональные архитектуры

- не только ООП :)

Для функц приложений есть свои шаблоны

Некоторые рассмотренные стили ... с функц подходом

* Mark Richards "Fundamentals of Software Architecture"
* Neal Ford "Software Architecture: The Hard Parts"

← актуальные книги

Архитектурные стили
Распределенные системы

→ сконцентрируемся на уровень системы (интеграция компонентов)

Что такое монолит?

- не сервис-ориентированное
- поставляется одним артефактом
- исп-ся в одном пр-ве
- комп. исп-ся на единой платформе
- ....

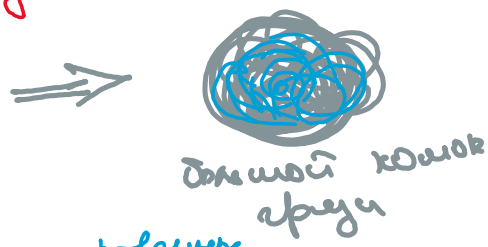
что-то большое и целое

* Это НЕ всегда плохо
  - Android IOS приложения
  - текст-редакторы и т.д.
  - утилиты...

\+ проще в разработке
\+ проще разворачивать
\+ меньше усилий для настройки
\+ проще поддерживать консистентность
\+ проще поддерживать

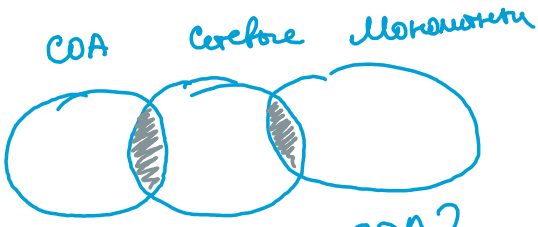
\- внутренняя сложность (при увеличении объёма кода)
\- люди не называют его "целиком"
\- превращается в "большой комок грязи"

- долго собирается
- доставка изменений и развертывание обновлений - длительный процесс
- масштабирование сложнее
- привязка к одному стеку технологий
- надёжность падает



большой комок грязи

→ сервис-ориентированные не обязательно сетевые



Что же такое SOA?

в следующий раз :)