

# 4. Модели на разпределена софтуерна архитектура

Васил Георгиев



[ci.fmi.uni-sofia.bg/](http://ci.fmi.uni-sofia.bg/)



[v.georgiev@fmi.uni-sofia.bg](mailto:v.georgiev@fmi.uni-sofia.bg)

# Съдържание

- Модели софтуерна архитектура
- Спецификации с **UML**
- Структурни и функционални диаграми
- Модели на изгледи
- Спецификации с **ADL**

# Модели софтуерна архитектура

- ✦ Софтуерната архитектура представя – т.е. моделира – програмния проект (процес на обслужване) като съставен т.е. разпределен процес от софтуерни компоненти
- ✦ моделирането на РСА е първата и най-важна фаза на проектиране, настройка, тестване, разгръщане и документация на разпределени среди за обслужване
- ✦ моделът на дадена софтуерна архитектура описва
  - ✦ декомпозицията на процеса на компоненти
  - ✦ функционалната им композиция
  - ✦ прилагания архитектурен стил – напр. процедурен, обектен, потоков (**data flow**), йерархичен или не-йерархичен, информационен (**data centric**), интерактивен (**interaction oriented**), базиран на изгледи (**views**) и др.
  - ✦ качествените (нефункционалните) атрибути на услугата – QoS

# Представяне на софтуерните модели

- Използват се графи и техни разширения
- описанието е чрез диаграми или техни текстови еквиваленти
- цели на описанието са
  - визуализация
  - спецификация
  - конструиране
  - документация
- ☞ следователно обикновено моделът включва мн. повече от една диаграма
- описанието (моделирането) стартира от по-упростените концепции на бизнес-модела или потребителския сценарий
  - напр. едномерен модел с блокова диаграма (ненасочен граф) – 4.4
- за по пълно функционално и нефункционално описание на проекта се прилагат многомерни модели
  - напр. «4+1» модели, включващи
    - логически изглед
    - изглед процеси
    - изглед проектиране
    - физически изглед
    - потребителски интерфейсни изгледи

# UML-модели на СА

- ✦ използва се за ОО-спецификация, анализ, проектиране и документиране на софтуерни проекти
- ✦ спецификациите са в две групи диаграми:
- ✦ структурни диаграми – **статично** описание (изреждане) на елементите в системата
  - ✦ йерархична библиотека класове
  - ✦ статични връзки между класовете
    - ✦ наследяване (“is α”)
    - ✦ асоциация (“uses α”)
    - ✦ агрегация (“has α”)
    - ✦ обмен (method invocation)
- ✦ функционални (**behavioral**) диаграми – **динамично** описание функциите (“поведението”) на инстанциите на класовете (т.е. обектите) с диаграми на
  - ✦ интеракцията,
  - ✦ колаборацията,
  - ✦ акцията и
  - ✦ конкурентността между обектите
- ✦ UML диаграмите могат да се транслират до HLL с общо приложение



# Структурни UML диаграми

Class	Изброяване и статични връзки между класовете (независещи от взаимодействието им по вр. на изпълнение)
Object	Извлечение от клас диаграмата за обектите и тяхното взаимодействие в определени специфични моменти от изпълнението на системата
Composite	<b>Диаграма на съставни структури</b> – описание на структурата на даден компонент като съставлящи го класове и компонентните интерфейси
Component	Описание на системата като структура от компоненти, интерфейсите между тях, и общите системни интерфейси
Package	Йерархична пакетна структура на организацията <b>класовете</b> в директории (т.е. групирани файлове) – пакети от класове и пакети от пакети
Deployment	<b>Диаграма на разгръщането</b> - описание на изпълнителната инфраструктура: сървери, изпълняващи <b>компонентите</b> , системно осигуряване и мидълуер, интерфейси и протоколи, вътрешна и външна мрежова свързаност

# Функционални UML диаграми...

Use case	<b>Диаграма на случай на употреба</b> – потребителските сценарии на заявки към системата и техните реакции – за описание на <b>функционалните и нефункционалните изисквания</b> към системата
Activity	<b>Диаграма на дейностите</b> – описание на контролния и контекстния обмен между класовете като мрежа от акции, които системата изпълнява за да осъществи реакциите по потребителския сценарий – <b>оркестрация на акциите</b>
State Machine	<b>Диаграма на машина на състоянията</b> – описание на жизнения цикъл на <b>обектите като машина на състоянията</b> – диаграми на състоянията и преходите (активни вътрешно-обусловени и реактивни външнообусловени преходи)

## ... функционални UML диаграми

Inter-action Overview	Диаграма за преглед на взаимодействието – описва <b>потока команди</b> между обектите (control flow) и е комбинация от Action и Sequence диаграмите
Sequence	Диаграма на последователност - <b>нареден (т.е. времеви) списък от съобщенията</b> между обектите
Communication	Аналогично на Sequence диаграмата, но структурирана като като <b>комуникационни канали</b> , които съдържат определен брой последователности
Time Sequence	Времево описание на преходите между вътрешните състояния на обектите и на различимите външни събития (от потребителския сценарий) като последователност от съобщения



# Class диаграми

- най-разпространеното описание при всеки модел
- статично изброяване на съставните блокове на модела като **класове**
- задава «**речника**» на модела в съответствие с проблемната област
- класовете се описват с техните атрибути
  - тип
  - интерфейс
  - методи
  - свойства
- достъпността (видимостта) на атрибутите се описва като
  - `public`
  - `private`
  - `protected`
  - `default`
- описва се и отношенията между класовете – наследяване, асоциация, агрегация (чрез дъги)
  - а също и мощността на тези отношения – 1:1, 1:много и т.н. (чрез маркировки в края на дъгите)

# Class диаграма - пример

- фиг. 4.10
- система за потребителски заявки
- наследственост (стрелка към родителя/базовия клас)
- агрегация (ромб към корена)
- асоциация (нейерархична дъга)
- маркировка на мощността в двата края на дъгите



# Објект диаграми

- ✦ извлича се от клас-диаграмата
- ✦ описва обектите като инстанции на класовете т.е. примерно подмножество обекти за дадена клас-диаграма конкретен момент на работа на системата
- ✦ пример – фиг. 4.11



# Composite Structure диаграми

- ✦ описва връзката между обектите (runtime), с което разширява “речника” на модела
- ✦ обектите и връзката се аотират с етикети – съответно на ролята (бизнес- или функционална логика) и отношението им (“колаборацията”)
- ✦ пример – фиг. 4.12





# Component диаграми

- компонентите са изпълними **SW**-модули за многократно използване при проектиране, които се представят със своя интерфейс
- в **UML** те са със скрита структура (черна кутия) [но при различните технологии се прилагат и компоненти тип “сива” и “стъклена кутия”]
- напр.
  - **jar** в компонентната библиотека **JavaBean**
  - **dll** в **.Net**
- компонентната диаграма представя съответствието между изискваните (полукръгче) и имплементираниите (кръгче) интерфейси – **фиг. 4.13**
- компонентите в даден проект може да са готови – **COTS** – и специфични

# Packet и Deployment диаграми

- фиг. 4.14.1
- фиг. 4.14.2



# Use case диаграми

- описва потребителските сценарии на приложение на системата като граф от актори, случаи на употреба (потребителски функции) и връзките между тях
- акторите са крайни потребители или други системи, приложения и устройства
- случаите (**Use Cases**) са комплексни функционални модули от разпределеното приложение/проекта, които описват отделни стъпки от цялостната бизнес-логика
- описанието на случаите се допълва в други диаграми с пред- и след-условията на изпълнението им като последователности от стъпките на общото приложение при конкретно негово изпълнение
- връзките между сценариите (фиг. 4.15) се маркират с
  - `<<include>>` от случай, който използва друг случай за изпълнение на дадена функция (насочена дъга)
  - `<<extend>>` от случай, който извиква друг такъв за изпълнение на функция по изключение (т.е. като опция, която се изпълнява само по изключение)
- диаграмите на случаите на употреба са основа на описанието и [началните] им версии се използват за основа на структурните и **sequence** диаграмите

# Activity диаграми

- ✦ описват проекта като **потоков (workflow)** бизнес процес, състоящ се от дейности – **activities**
- ✦ дейностите капсулират
  - ✦ логиката на взимането на решение
  - ✦ конкурентното изпълнение на функции
  - ✦ обработката на изключения
  - ✦ прекратяването на процеса (**termination**)
- ✦ потоковата **activity** диаграма (фиг. 4.16) се състои от
  - ✦ една начална точка и поне една крайна точка (плътен кръг и ограден кръг)
  - ✦ точките на решаване (означават се с ромбче)
  - ✦ другите дейности (заоблен правоъгълник)
  - ✦ конкурентното разделяне и събиране на потоците (дебела черта); **N.B.** – събирането на два и повече потока се счита за синхронизатор (следващите го дейности не могат да стартират без завършване на *всички* предхождащи го)
  - ✦ събития (**events** - опция) – представят обмена на съобщения (**signals**) между конкурентните акции (насочени многоъгълници с етикети)



# State Machine диаграми

- обикновено представят състоянието на обслужващите устройства или софтуерните модули в проекта – набор от състоянията им и преходите между тях
- логиката на състоянията е реактивна – т.е. базира се на външни събития (**events**)
- състоянията се описват с блок, съдържащ
  - име,
  - списък променливи и
  - **activity**
- State Machine диаграмата (фиг. 4.17) се състои от
  - една начална точка и поне една крайна точка (плътен кръг и ограден кръг)
  - насочени маркирани дъги на преходите
  - състоянията, които може да са комплексни състояния, съставени от допълващи State Machine диаграми

# Interaction Overview, последователности и времеви диаграми

- ✦ диаграмите за преглед на взаимодействието се състоят от кадри (frames), които представляват други диаграми на проекта, маркирани с указател (reference) или със самите диаграми, маркирани с тип – напр. sd, cd, ad
- ✦ дъгите отразяват контролния поток на взаимодействието - фиг. 4.18.1
- ✦ sequence диаграмите отразяват относителната последователност от контролни съобщения между обектите – фиг. 4.18.2
- ✦ времевата диаграма описва графика на състоянията от машината на състоянията - прилага се за RT приложения и системи – RTOS, ES (4.18.3)

# Модел на изгледи

- 4+1 моделиране – представя РСА с 4 основни изгледа и един допълнителен – логически, развоен, процесен и физически + сценарий на приложение/функциониране, който често се придружава и от изглед на потребителските интерфейси – фиг. 4.19
- Сценарният изглед и асоциираният с него интерфейсен изглед описват потребителските функции на приложението както и основните нефункционелни изисквания
  - произтича от потребителското задание
  - в UML се специфицира с диаграма на потребителските случаи (4.15)
- Логическият изглед описва декомпозицията на разпределеното приложение с оглед на реализираните функции
  - представя основните блокове или компоненти
  - в UML се специфицира с клас-диаграма (статична), допълнена с една или повече динамични диаграми – най-често последователности

# Развоен, процесен и физически изглед

- Развойният изглед и асоциираният с него интерфейсен изглед описват потребителските функции на приложението както и основните нефункционални изисквания
  - произтича от потребителското задание
  - в UML се специфицира с диаграма на потребителските случаи (4.15)
- Процесният изглед описва декомпозицията на разпределеното приложение с оглед на реализираните функции
  - представя основните блокове или компоненти
  - в UML се специфицира с клас-диаграма (статична), допълнена с една или повече динамични диаграми – най-често последователности или на дейностите (4.20.1)
- Физическият изглед описва цялата РСА на платформата + приложението – инсталация, конфигурация, разгръщане
  - компонентите са на ниво процесори или поне процеси
  - връзките между тях са на ниво комуникационни канали
  - представя нанасянето (или картирането – mapping) на компонентите от развойния изглед върху инфраструктурните възли (4.20.2)



# Потребителски интерфейс изглед



# ADL

- Architectural Description Language – графична спецификация на модели на разпределена софтуерна архитектура
- свободно разпространявана среда за спецификация на ADL-модели AcmeStudio (<http://www.cs.cmu.edu/~acme/AcmeStudio/index.html>) с автоматична генерация на Java и C++

The screenshot displays the AcmeStudio Architecture Design Environment. The main window shows a diagram of a distributed software architecture. A central yellow cylinder represents the 'Target\_System'. Above it, a 'Translator' (green oval) is connected to a 'Gauge Bus' (grey bar) and a 'Probe Bus' (grey bar). The 'Gauge Bus' is connected to two green circles labeled 'G1' and 'G2'. The 'Probe Bus' is connected to a yellow rectangle labeled 'ModelManagerT'. A red triangle with an exclamation mark is visible in the top left of the diagram area. The interface includes a menu bar (File, Edit, View, Navigate, Search, Project, Family, Design, Window, Help), a toolbar, a Navigator pane on the left showing a project tree, a Palette on the right with various components and connectors, and a Properties pane at the bottom showing the 'Basic' properties of the selected 'RainbowSys' component.

AcmeStudio - Rainbow/RainbowSys.acme - AcmeStudio Architecture Design Environment

File Edit View Navigate Search Project Family Design Window Help

Navigator \*RainbowSys.acme

Palette

- Select
- Connect
- Default
- RainbowFam
- Components
  - RainbowT
  - TargetSysT
  - ModelManagerT
  - GaugeT
- Connectors
  - GaugeBusT
  - TranslationT
  - ProbeBusT
- Ports
  - SystemChangePortT
  - ModelUpdatePortT
  - EffectorDataT
  - GaugeInT
  - ChangePortT
  - ProbeDataT

Overview Acme Source RainbowSys

Properties Tasks Problems

Basic

Name: RainbowSys

Description:

4. Модели софтуерна архитектура

ФМИ/СУ \* СИ \* PCA

22