|  |  |
| --- | --- |
| **4 - Модели на софтуерната архитектура. Спецификация с UML и ADL.**   1. **Модели софтуерна архитектура**   Софтуерната архитектура представя (моделира) програмния проект (процес на обслужване) като съставен, т.е. разпределен процес от софтуерни компоненти.  Моделирането на разпределена софтуерна архитектура 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, качествените (нефункционалните) атрибути на услугата – QoS.  За представяне се използват графи и техните разширения. Описанието е чрез диаграми или текстови еквиваленти. Целта на описанието е за визуализация, спецификация, конструриране и документиране => обикновено моделът включва много повече от една диаграма. Описанието (моделирането) стартира от по-упростени концепции на бизнес-модела или потребителския сценарий. Напр. едномерен модел с блокова диаграма (ненасочен граф) – фиг. 4.4(разглеждаме диаграма за бизнес приложение за електронна търговия, която представлява неориентиран граф). 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 интерфейс изгледи.   1. **Спецификации с UML**   UML е средство за декомпозиране на проекта(софтуерната архитектура) в обектен модел. Този модел се състои от множество диаграми, представящи различни аспекти на проекта.  UML-модели на софтуерна архитектура се използват за ОО-спецификация, анализ, проектиране и документиране на софтуерни проекти. Спецификациите са в **2 групи диаграми:**  1**)структурни диаграми** – **статично** описание(изреждане) на елементите в системата (йерархична библиотека класове и статични връзки между класове като наследяване („is a”), асоциация (“uses a”), агрегация(“has a”), обмен(method invocation)).  2) **функционални (behavioral) диаграми** – **динамично** описание на функциите(„поведението”) на инстанциите на класовете (т.е. обектите) с диаграми на интеракцията, колаборацията, акцията и конкурентноспособността между обектите.  UML диаграмите могат да се транслират до HLL с общо приложение   1. **Структурни и функционални диаграми**   **Структурни UML диаграми:**  **Class** – изброяване и статични връзки между класовете (независещи от взаимодействието им по време на изпълнение). Те са най-разпространеното описание при всеки модел. При тях се прави статично изброяване на съставните блокове на модела като **класове**. Задава се **„речника”** на модела в съответствие с проблемната област. Класовете се описват с техните атрибути, които са име, интерфейс, методи, свойства. Достъпността (видимостта) на атрибутите се описва като public, private, protected, default. Описват се и отношенията между класовете – наследяване, асоциация, агрегация (чрез дъги), а също и мощността на тези отношение: 1:1, 1:много и т.н. (чрез маркировки в края на дъгите). Пример: фиг. 4.10 – система за потребителски заявки. Всяко блокче съдържа име на класа, атрибути и методи.Изполсват се още ромб към корена – агрегация, стрелка към базовия клас – наследяване, нейерархична дъга – асоциация, които показват връзките между класовете. Слага се и маркировка на мощностите в двата края на дъгите.  **Object** – извлечение от клас диаграмата. Описва обектите като инстанции на класовете, т.е. примерно подмножество обекти за дадена клас-диаграма, и тяхното взаимодействие в определени специфични(конкретни) моменти от изпълнението на системата.Пример:фиг. 4.11  **Composite** – Диаграма на съставните структури – описание на структурата на даден компонент като съставящи го класове и компонентните интерфейси. Задава „речника” – логическите операции, които съществуват. Описва връзката между обектите (runtime), с което разширява „речника” на модела. Обектите и връзките се анотират с етикети – съответно на ролята( бизнес или функционална логика) и отношението им („колаборацията”). Пример:фиг.4.12  **Component** – описание на системата като структура от компоненти, интерфейсите между тях, и общите системни интерфейси. Компонентите са изпълними software – модули за многократно използване при проектирани, 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” или „стъклена кутия”], напр. jar в компонентната библиотека JavaBean или dll в .NET. Компонентната диаграма представя съответствието между изискваните(полукръгче) и имплементираните(кръгче) интерфейси – фиг. 4.13. Компонентите в даден проект може да са готови – COTS – и специфични.  **Package** – Йерархична пакетна структура на организация на класовете в директории(т.е. групирани файлове) – пакети от класове и пакети от пакети – фиг.4.14.1  **Deplyment** – Диаграма на разгръщането – описание на изпълнителната инфраструктура: сървери, изпълняващи компонентите, системно осигуряване и мидълуер, интерфейси и протоколи, вътрешна и външна мрежова свързаност – фиг.4.14.2  **Функционални UML диаграми:**  **Use case** – Диаграма на случайна употреба – описват потребителските сценарии на заявки към системата и техните реакции като граф от актьори, случаи на употреба(потребителски функции) и връзките между тях– за описание на **функционалните и нефункционалните изисквания** към системата. Акторите са крайни потребители или други системи, приложения и устройства. Случаите (uses cases) са комплексни функционални модули от разпределеното приложение/проект, който описва отделни стъпки от цялостната бизнес-логика. Описанието на случаите се допълва в други диаграми с пред- или 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 стрелка), <<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 случайна употреба са основа на описанието и [началните] им версии се използват за основа на структурните и sequence диаграмите.  **Activity** – Диаграма на дейностите – описание на контролния и контекстния обмен между класовете като мрежа от акции, които системата изпълнява, за да осъществи реакциите по потребителския сценарий – **оркестрация на акциите.** Този вид диаграми описват проекта като **потоков** (workflow) бизнез процес, състоящ се от дейности – activities. Дейностите капсулират логиката на взимането на решение, конкурентното изпълнение на функции, обработката на изключения и 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), други дейности (заоблен правоъгълник), конкурентно разпределяне и събиране на потоците (дебела черта) (N.B. – събирането на два и повече потока се счита за синхронизатор (следващите го дейности не могат да се страртират без завършване на всички предхождащи го) ) , събития (events-опция) – представят обмена на съобщенията (signals) между конкурентните акции (насочени многоъгълници с етикети).  **State Machine** – Диаграма на машина на състоянията – описание на жизнения цикъл на **обектите като машина на състоянията** – диаграми на състоянията и преходите(активни вътрешно обусловени и реактивни външно обусловени преходи). Състоянията се описват с блок, съдържащ име, списък променливи и activity. Логиката на състоянията е реактивна, т.е. се базира на външни събития (events). Диаграмата на 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 се State Machine диаграми.  **Interaction Overview** – Диаграма за преглед на взаимодействието – описва **потока команди** между обектите (control flow) и е комбинация от Action и Sequence диаграмите. Този вид диаграми 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. На фиг.4.18.1 дъгите отразяват контролния поток на взаимодействието.  **Sequence** – Диаграма на последователност – **нареден (т.е. времеви) списък от съобщенията** между обектите. Те отразяват относителната последователност от контролни съобщения между обектите – фиг 4.18.2  **Communication** – аналогично на Sequence диаграмата, но структурирана като **комуникационни канали**, 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. Прилага се за RT приложения или системи – RTOS, ES (фиг.4.18.3)   1. **Модели на изгледи**   „4+1” моделиране – представя разпределена софтуерна архитектура с 4 основни изгледа и един допълнителен – логически, развоен, процесен и физически + сценарий на функциониране, който често се придружава и от изглед на потребителския интерфейс – фиг.4.19.  **Сценарния изглед** и асоциираният с него интерфейсен изглед описват потребителските функции на приложението, както и основните нефункционални изисквания. Той произтича от потребителското задание, а в UML се специфицира с диаграма на потребителските случаи(use case диаграми).  **Логическия изглед** описва декомпозицията на разпределеното приложение с оглед на реализираните функции. Този изглед представя основните блокове или компоненти. В UML се специфицира клас-диаграма (статична), допълнена с една или повече динамични диаграми – най-често последователностни.  **Развойният изглед** и асодиираният с него интерфейсен изглед описват потребителските функции на приловението както и основните нефункционални изисквания. Този изглед произтича от потребителското задание. В UML се специфицира с диаграма на потребителските случаи (use case).  **Процесният изглед** описва декомпозицията на разпределеното приложение с оглед на реализираните функции. Този изглед представя основните блокове и компоненти. В UML се прецифицира с клас диаграма (статична), допълнена с една или повече динамични диаграми – най-често последователностни или на дейностите(фиг.4.20.1).  Физическият изглед описва цялата разпределена софтуерна архитектура на платформата + приложението – инсталация, конфигурация, разгръщане, Компонентите са на ниво процесори или поне процеси. Връзките между тях са на ниво комуникационни канали. Този изглед представя бабасябети (или картирането – mapping) на компонентите от развойния изглед върху инфраструктурните възли (фиг.4.20.2)   1. **Спецификации с ADL**(Architectural Description Language – графична спецификация на модели за разпределена софтуерна архитектура). Съществува свободно разпространявана среда за спецификации на ADL – модели AcmeStudio с автоматична генерация на Java и |  |