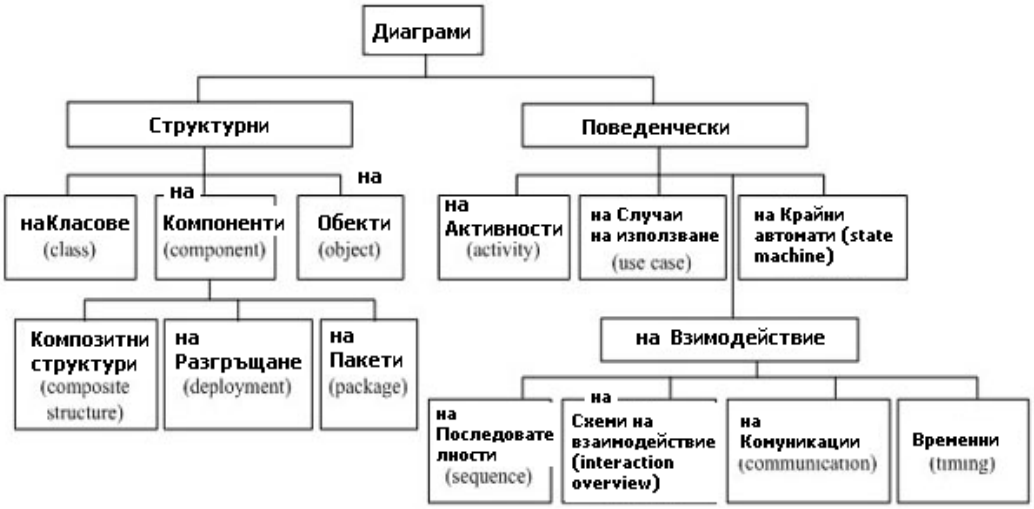
Упражнение 13-14. UML диаграми на компоненти, диаграми на разполагане и диаграми на пакети

Цел: Запознаване с предназначението, графичната нотация и процеса на построяване на UML Component Diagrams, UML Deployment Diagrams и UML Package Diagrams.

1. **Диаграми на компоненти**

Същност и предназначение на Диаграмите на компоненти

Да си припомним типовете диаграми в UML 2.0 и мястото на Диаграмите на компоненти, разполагане и пакети:



Това е структурна диаграма. Основната цел на тази диаграма (component diagram) е да покаже архитектурата на компонентите на системата и зависимостите между тези компоненти.

Това е структурна UML диаграма. Тя помага да се моделира физическия аспект на една Object-Oriented софтуерна система.

Всички диаграми, с изключение на тази и на разгръщане касаят концептуалните и логически аспекти на построяване на модела на системата. Особеността на логическото представяне се заключава в това, че то оперира с понятия, които нямат материален израз. С други думи, различните елементи на логическото представяне, такива като класове, асоциации, състояния, съобщения, не съществуват материално или физически. Те само отразяват разбирането за статичната структура на тази или друга система или динамичните аспекти на нейното поведение.

За създаване на конкретната физическа система е необходимо да се реализират всички елементи на логическото представяне в конкретни материални същности. Пълният проект на една програмна система представлява съвокупност от моделите на логическото и физическото представяне, които трябва да бъдат съгласувани помежду си. В езика UML за физическото представяне на моделите на системата се използват така наречените диаграми на реализацията, а именно: диаграма на компоненти и диаграма на разгръщане.

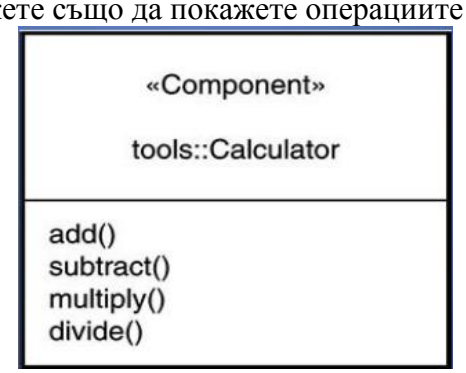
Диаграмата на компоненти позволява да се създаде архитектурата на разработваната система, описвайки компонентите на системата (в ролята на каквито могат да бъдат: изходен, бинарен и изпълним код, например: изпълним файл, динамична библиотека, Web-страница, текстов файл или файл-справка, файл на бази от данни, файл с изходен текст на програма, файл със скрипт и др.), заедно с интерфейсите и зависимостите между тях.

Основни графични елементи на тези диаграми са компонентите, интерфейсите и зависимостите между компонентите.

Какво е компонент?

Понятието компонент в UML се явява много мощно и еднозначно определение за него не съществува. Неоднозначността възниква не толкова от различните гледни точки на изследователите,

а по скоро във връзка с разпространението на различни компонентни технологии (JavaBeans, EJB, CORBA, DCOM, .Net) и средства за програмиране, използващи това понятие по различен начин.
Дефиниция 1: Компонентът е модулна част на системата, това е софтуерна имплементация на един или повече класа, намираща се на даден компютър (тоест, физически съществува част на ПС, а не нещо което е във въображението на един анализатор). Един компонент предоставя (осигурява) интерфейси на други компоненти.
Дефиниция 2: Компонентите (components) – това са независими модули на ПО, скриващи своята реализация и взаимодействащи помежду си с интерфейси.
Първоначално, в UML 1.1 компонентите се отъждествяваха с изпълними елементи (изпълним файл, динамична библиотека) , даннови файлове, таблици, документи. В действителност, проектантите често ги класифицират на компоненти на разполагане (deployment components), работни компоненти (work product components), и изпълними компоненти (execution components).
Това понятие еволюира във времето и в UML 2, компонентите се наричат артефакти – т.е части информация, които системата използва или произвежда.
Един компонент, от друга страна, дефинира функционалност на системата. Точно както компонентът е имплементация на един или повече класове, артефактът (ако е изпълним) е имплементация на един компонент и ако компонентът е член на пакет, можете да използвате за префикс пред името на компонент името на пакета. Можете също да покажете операциите на компонент в отделен панел.



Независимо от еволюцията на понятието компонент и замяната му с "артефакт", ние ще продължим да използваме понятието "компонент" вместо артефакт, като ще влагаме разбирането на UML 2 за него.
Изгледи на Компонент...
Както системата, така и системният компонент има

- вътрешен/структурен изглед и
- външен/функционален/поведенчески изглед.

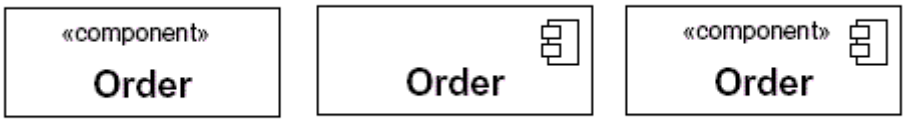
Външният/функционален/поведенчески изглед на компонента се специфицира с множество предоставени (provided) от компонента интерфейси и с множество изисквани, т.е използвани (required) от компонента интерфейси.
Графично представяне на компонент
За графическото представяне на компонент се използва специален символ (икона) – UML 1.x нотация: правоъгълник с 2 малки правоъгълника на лявата страна. Вътре в големия се записва името на компонента, а възможно и допълнителна информация. В този случай двата малки правоъгълника обозначават съответно: горния обозначава данни, които има този компонент, а долния - операциите или методите, реализирани с компонента.

Друг начин за указване на стреотипа на различните видове компоненти е чрез текст пред името на компонента:

- <<file>> - (файл) – произволен физически файл, например текстов файл и др.
- <<executable>> (изпълним файл - .exe)
- <<document>> (документ, Web страница).
- <<library>> (реферирам файл - .dll).
- <<source>> (източник, например изходен код, т.е source code file).
- <<table>> (таблица, таблица на БД – database table).

Следващата Figure 2 показва 3 други начини за изображение на компонент според UML 2 specification.

!!! Не забравяйте да поставите в този случай стереотип "component" или иконата на компонент в правоъгълника, иначе – това ще означава клас.

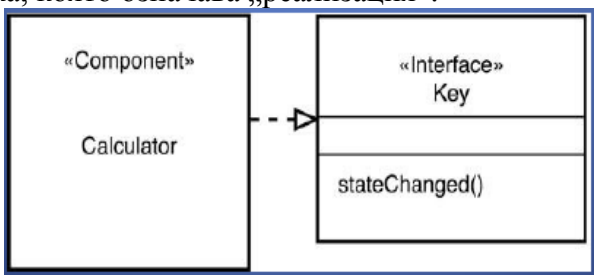


Графично представяне на интерфейс

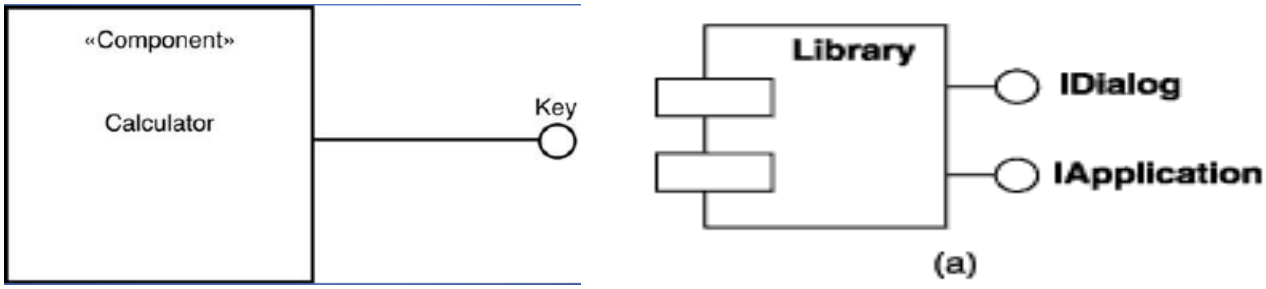
Един интерфейс е множество от свързани операции или услуги.

Представяне на предоставени (provided) от компонента интерфейси: Тези интерфейси могат да се представят по 2 начина:

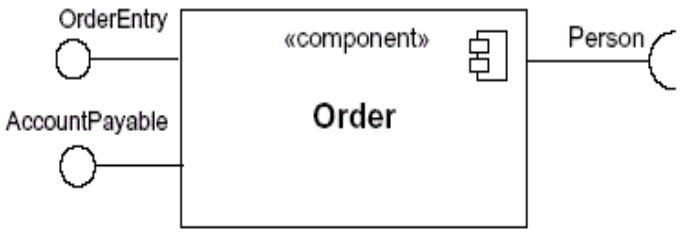
Първият: Интерфейсът е като правоъгълник, който съдържа информация на интерфейсите, които компонетът предоставя и използва. Предоставеният интерфейс се свързва с компонента с пунктирана линия и куха отворена стрелка, която означава „реализация”:



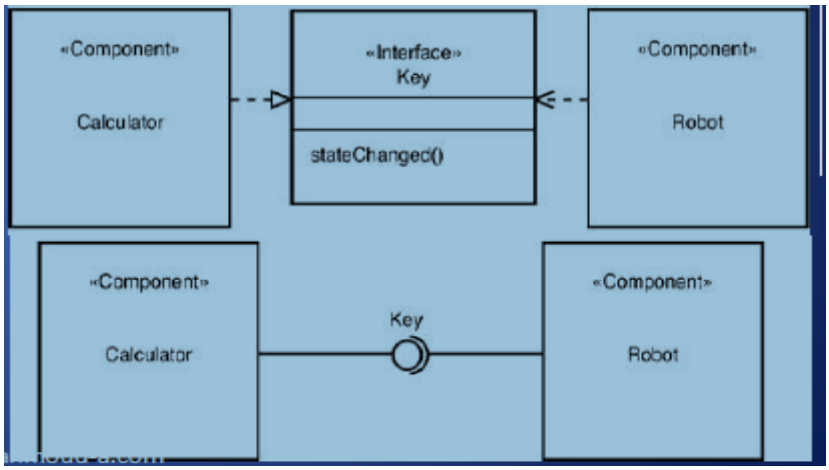
Вторият начин е - представяне на предоставения интерфейс като малък кръг, свързан с компонента със солидна линия (“Lollipop” символ, т.е “близалка”):



Името на интерфейса, се препоръчва да започва с "I", и се записва до кръгчето. Фигурата по-долу показва, че компонент Order предоставя 2 интерфейса: OrderEntry и AccountPayable и зависи от друг компонент, който предоставя Person interface, т.е Order изисква/използва Person interface. Използваните интерфейси се обозначат с отворена чашка (socket):



Един компонент може да достъпва операциите на друг компонент, чрез неговия интерфейс.
Зависимост: Връзка между един компонент и интерфейс (на друг компонент), чрез който се достъпва другия компонент е именно връзка от тип „зависимост". Представя се чрез пунктирана линия с отворена стрелка:



В какво се изразява независимостта на компонентите? В съществено различната функционалност, която отделните модули реализират по отношение на системата.

2. Диаграми на разполагане (разгръщане) в UML

Същност и предназначение на диаграмите на разгръщане
Диаграмата на разгръщане изобразява статичния изглед на конфигурацията на хардуерните възли по време на изпълнение на софтуерни компоненти върху тези възли. UML диаграми на разгръщане показват хардуера на вашата система, софтуерът, който е инсталиран на този хардуер и мидълуер (middleware), който се използва за свързване на разнородни машини една с друга.
Вие създавате UML модел на разполагане (deployment model) с цел да:

1. изследвате въпросите, свързани с реалното инсталирането на вашата система,
2. изследване на зависимостите на вашата система от други системи, които в момента са в експлоатация, или са планирани за разработка,
3. изобразите основната конфигурация за разполагане на бизнес приложение,
4. проектирате хардуерната и софтуерна конфигурация на вградена система, или
5. изобразите хардуерната /мрежовата инфраструктура на организация.

UML 2 вече дефинира формално едно устройството като възел, който изпълнява артефакти. (Помнете, че в UML 2, компонентът е класифициран като артефакт).
Представяне на Възел - Node: Указва се име на възела, а може да се добави и ключовата дума «Device», въпреки че това не е задължително необходимо.
Диаграмите на разгръщане са полезни, когато вашият софтуер е разположен на много машини, всяка от които има уникална конфигурация. Ето ще един пример:
Възли и Компоненти (Nodes, Components):
В UML 2, възлите (nodes) могат да бъдат също и Софтуерни Компоненти. Компонентите (Components), представляват софтуерни артефакти, като файлове, рамки или компоненти на областта (file, framework или domain component).

- Именовайте Възлите (Nodes) с описателни имена (client, Application Server, Database Server … и т.н).
- Моделирайте само важните Софтуерни Компоненти (Components).
- Използвайте постоянни (съгласувани) стереотипи за компоненти.
- Използвайте визуални стереотипи за възли (Nodes).

Зависимост (**Dependency**) и Комуникационни Асоциации (**Communication Associations**)

- Комуникационните асоциации, често наричани конектори (connections), се представят като линии, свързващи възлите.

- Зависимостта между компонентите се представя чрез пунктирани стрелки.

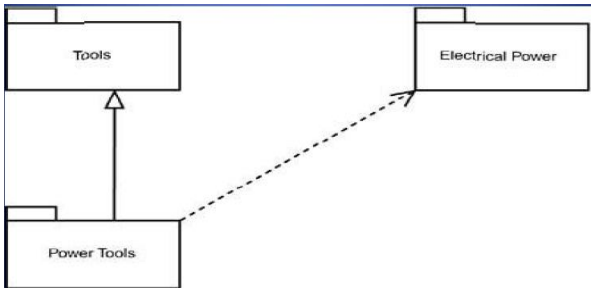
3. Диаграми на пакети в UML

Същност и предназначение на диаграмите на пакети

- Както подсказва името му, пакетът е предназначен за групиране (опаковане) на други UML диаграми (като класове или случаи употреба). Обградете група елементи с икона - папка и вече сте ги опаковали.
- Пакетът е UML конструкция, която позволява да организирате вашите диаграми.

Създаване на диаграма на пакети за да се:

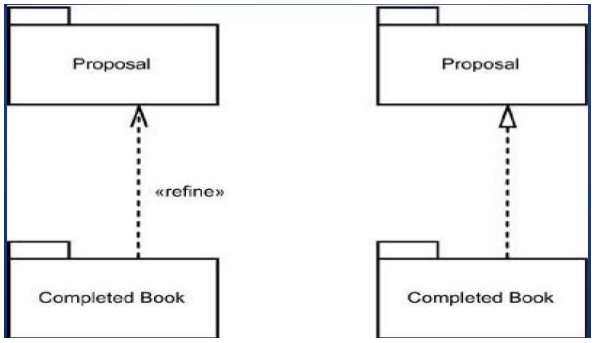
- Представи изглед на високо ниво на вашите изисквания
- Представи изглед на високо ниво на вашия проект
- Логическа модуларизация на една сложна програма



Връзки между пакети (Inter-package Relationships):
Пакетите могат да взаимодействат помежду си по един от 3 начина:
Един пакет може да **обобщава** друг, да **зависи** от друг или да **уточнява** друг пакет.

Обобщаване и Зависимост (Generalization & Dependency)

Уточняване (Refinement relationships) :
Уточняването е свързано с нивото на детайлизиранel. Един пакет уточнява друг, ако съдържа същите елементи, но с повече детайли.



Представяне на Уточняваща връзка (Refinement relationship) :
Има 2 начина за представяне:

Препоръки при създаване на Пакети (Packages) :

- Използвайте ясни и описващи имена на пакетите
- Използвайте пакети за да направите по-ясни и по-прости своите диаграми
- Пакетите трябва да са силно свързани (Cohesive)
- Зависимостите между пакетите трябва да отразяват зависимостите между компонентите, включени в пакетите (Когато един пакет зависи от друг, това означава че има една или повече връзки между съдържанията на двата пакета)
- Избягвайте циклични зависимости между пакети: Избягвайте ситуация в която един пакет A е зависещ от пакет B, който е зависещ от пакет C, а последният е зависещ от A, тоест в този случай имае цикъл: A→B→C→A. Тъй като са свързани помежду си, ще бъде трудно да бъдат тествани и поддържани. Наличието в даден момент на такава зависимост е индикация, че системата трябва да бъде преструктурирана, да се махнат от пакетите тези компоненти, които пречиняват циклична зависимост

Обобщение

Диаграми на компонентите (Component diagram)

• Предоставят изглед, свързан с имплементацията на системата (implementation view).
• Показват зависимостите между компонентите на системата.

Диаграми на разгръщане (Deployment diagram)

• Предоставят изглед, свързан с обкръжението на системата (environment view).
• Представят физическите връзки между софтуерните и хардуерните компоненти.
• Всеки възел представя хардуерна част (piece of hardware).

• Връзките показват комуникационни пътища.

Диаграма на пакети (Package diagram)

• Диаграмите се групират в логически-подредени пакети.
• Диаграмите на пакети показват връзки и зависимости между пакетите.
• Тези диаграми са важни за големи проекти

Виж. „Програмни спецификации . Ръководство за лабораторни упражнения" тема XI.

Литература

- Systems Analysis & Design with UML, 1nd Edition.
- Elements of UML 2.0 style.
- Software Engineering - Sams - Teach Yourself Uml In 24 Hours.