# 6. Йерархични, асинхронни и интерактивни модели на софтуерната архитектура

ВАСИЛ ГЕОРГИЕВ

http://www.fmi.uni-sofia.bg/bg/lecturers/ci/v_georgiev/

v.georgiev@fmi.uni-sofia.bg

# Съдържание

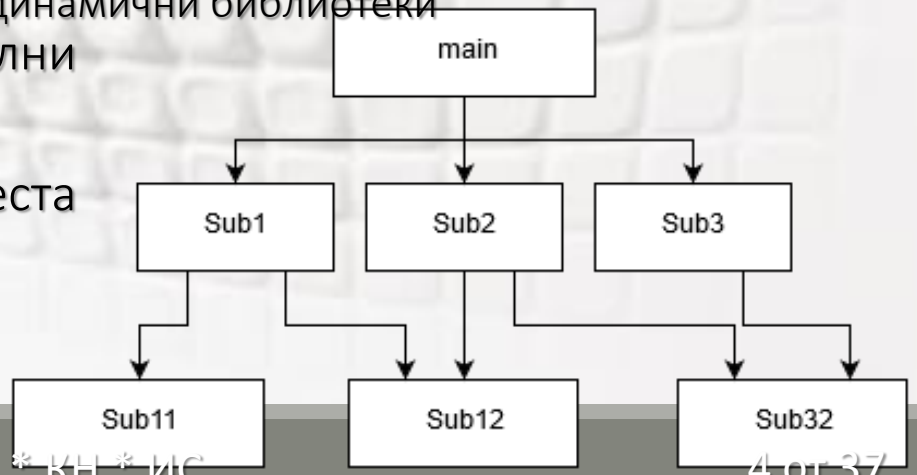
- Йерархични архитектури
  - подпрограми и Master-Slave
  - слоеста архитектура и виртуални машини
- Асинхронни архитектури
  - буферирани и небуферирани модели
- Интерактивни архитектури
  - модел-изглед-контролер – I и II
  - представяне-абстракция-контрол

# Йерархични архитектури

- декомпозират системата по управление на йерархични модули – т.е. функциите се групират по йерархичен принцип на няколко нива
- координацията обикновено е между модули от различни нива (вертикална свързаност) и се базира на явни (т.е. “заявка-отговор”) обръщения
- ниските нива функционират като услуги към непосредствените по-високи нива; услугите са имплементирани като функции и процедури или пакети от класове
- пълна прозрачност между нивата се постига при запазване на свързващите интерфейси, но имплементацията на услугите може да еволюира
- архитектурен модел на много ОС (Unix, MS .Net) и на протоколните стекове (TCP/IP); разслояване:
  - базови услуги – системните услуги се групират в модули за IO, транзакции, балансирано планиране на процеси, защита на информацията
  - междинен слой – “ядро” – поддържа проблемно-ориенетирана логика – бизнес приложения, числова обработка, информационна обработка, като представя интерфейси към колекции от базовите услуги
  - потребителски интерфейсен слой – напр. команден екран, графични контролни прозорци, Shell скрипт интерпретатор

# Йерархия с подпрограми (Main/Subroutine)

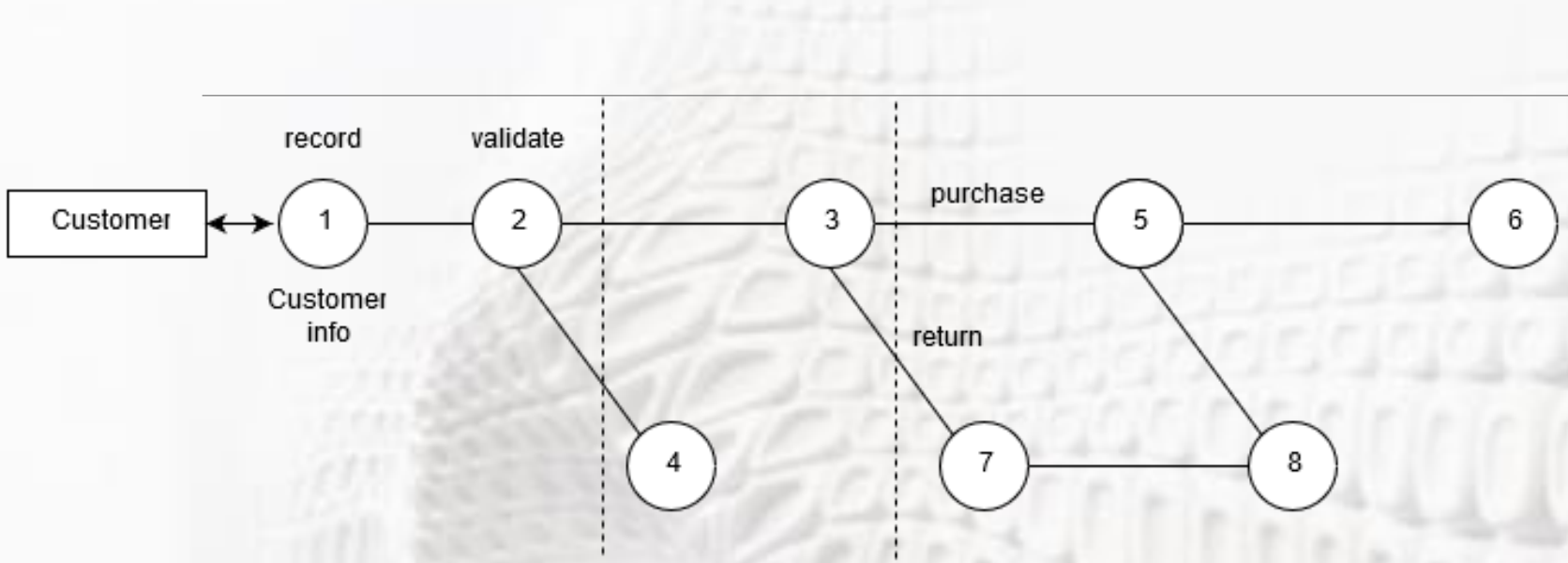
- традиционна архитектура, предхожда ОО, базира се на процедури със споделен достъп до данните (само частична капсулация)
- декомпозицията е по управление, като комплексната функционалност на приложението се разделя на на по-малки функционални групи – процедури и подпрограми – с цел тяхното споделяне между различни извикващи ги модули
- актуалните данни са параметри на обръщенията към изпълнителните функции и могат да се адресират по
  - указател – подпрограмата може променя техните стойности на същия адрес
  - стойност – подпрограмата получава стойностите като константи
  - име – подпрограмата използва като аргумент локалната стойност за съответното име (л-я 10.) – най-често това са локални имплементации на протоколи и други резидентни програми или динамични библиотеки
- главната програма управлява процеса на последователни обръщения към подпрограмите
- подпрограмите формират нефиксирана но ациклична слоеста йерархия – фиг. 6.4.

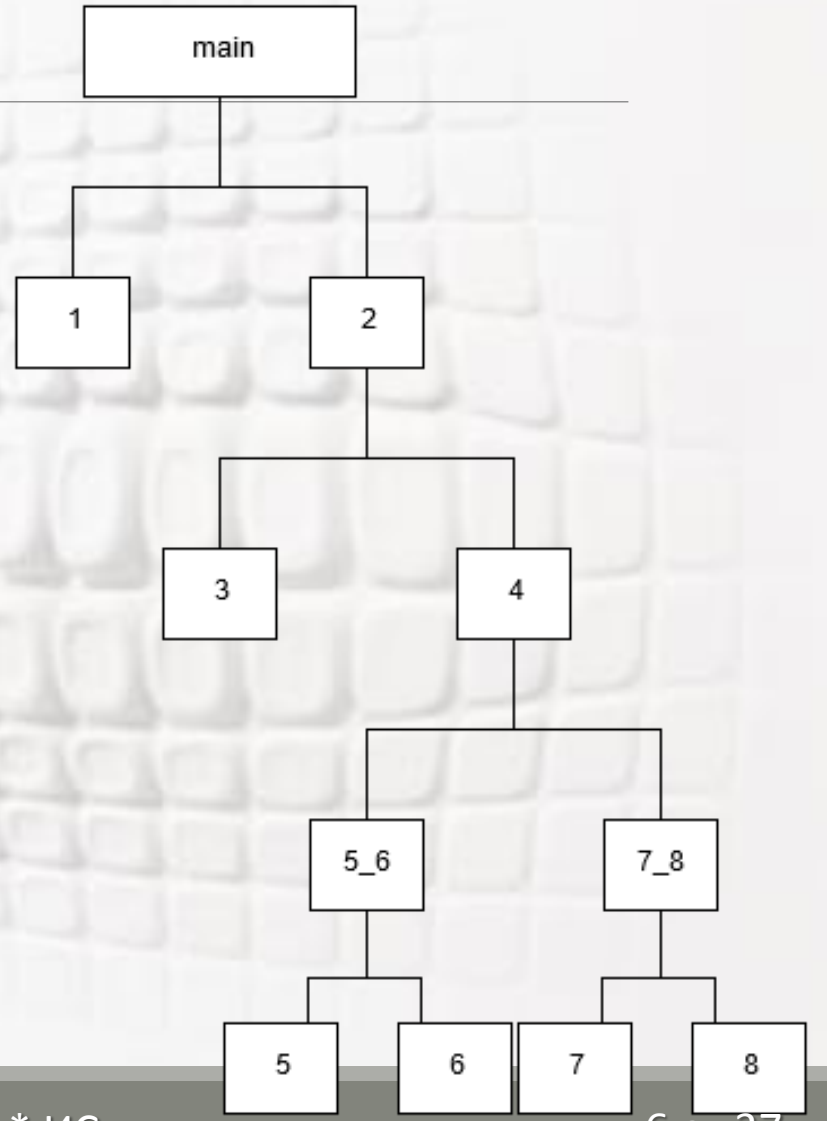


# Диаграми на MSub-архитектура

- потоковата диаграма се използва за начално моделиране на изискванията към системата
  - потокова диаграма на OPS (Order Processing) – местата отразяват обработката, а дъгите – преноса на данните – 6.5.1
  - възел 1 – регистрация на заявките; в. 2 – валидиране и отказ (в. 4) или предаване на заявката; в. 3 приема или отказва заявка (в зависимост от изпълнимостта); в. 5 променя стоковата наличност и предава за фактуриране на в. 6; в. 7 обработва правилата за отказ и предава на в. 8 за уведомление (примерно друга оферта)
- при анализа се идентифицират
  - трансформиращите възли – променят фórмата на входните данни (напр. XML) към вътрешен формат – обикн. възлите с един вход и един изход
  - транзактивните възли – обработват входящите данни и ги насочват към един или друг изходен поток или пък нямат изходящи дъги
- от потоковата диаграма се извлича блокова диаграма на архитектурата – която е съставена от контролни и диспечерски модули (подпрограми) – съответстващи респективно на трансформиращите и транзактивните възли на потоковата диаграма – 6.5.2

# Диаграми на MSub-архитектура – фиг. 6.5.1. и 6.5.2.

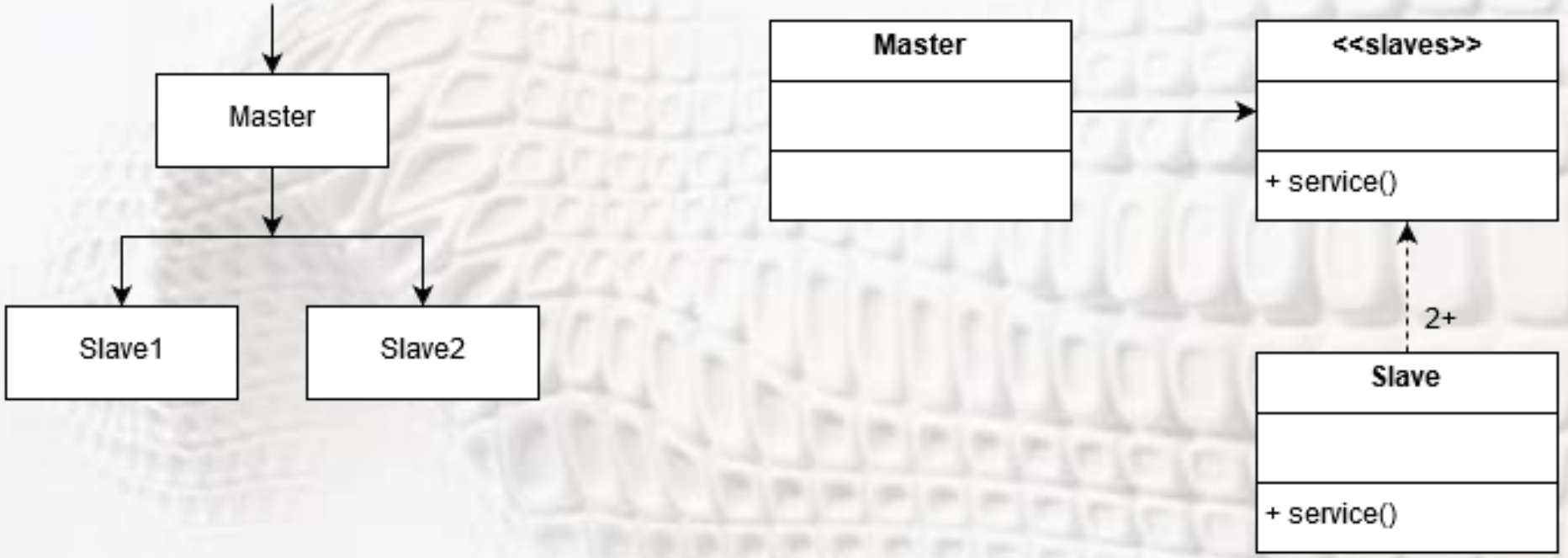


# Master/Slaves

- това е вариант на архитектурата с подпрограми, който е специализиран към поддържане на допълнителни нефункционални изисквания – най-ве4е
  - отказоустойчивост (fault tolerance) и надеждност
  - балансиране за ускорено изпълнение на заявките
- реализира се чрез репликиране на функционалните модули
- при M-S архитектурата задача на **M** е *алтернативно*:
  - отказоустойчивост: да оцени адекватността на паралелно обработените резултати от $S_i$ – съществуват протоколи за отказоустойчивост, идентифициращи грешните и верни резултати при ограничен брой на изпълнителните реплики
  - бързодействие: да извърши разпределяне на заявките прилагайки принципите за товарен баланс
- блок диаграма и клас диаграма на Master/Slaves архитектура – 6.7

# Диаграми на MS архитектура – фиг. 6.7.



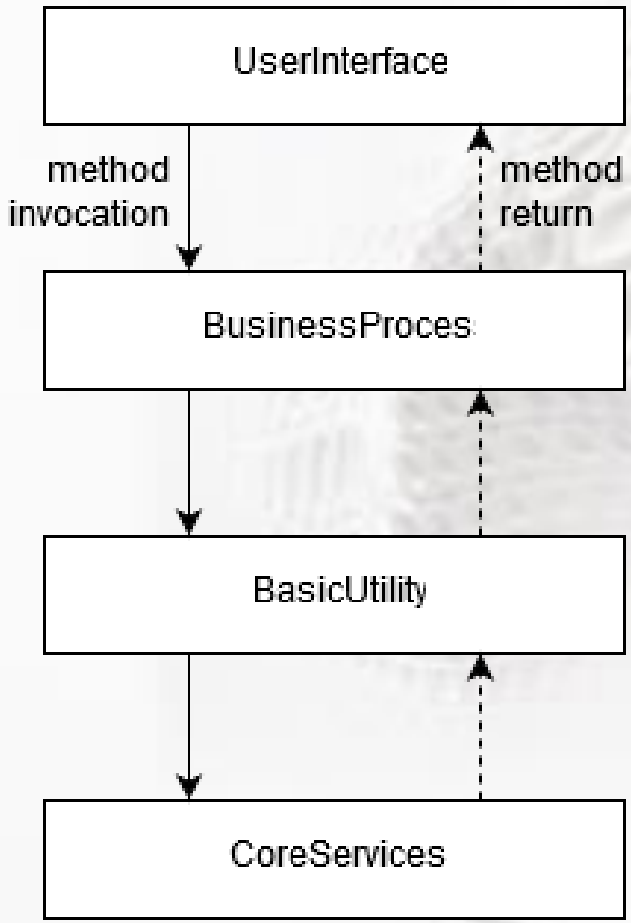
# Обхват на подпрограмните архитектури

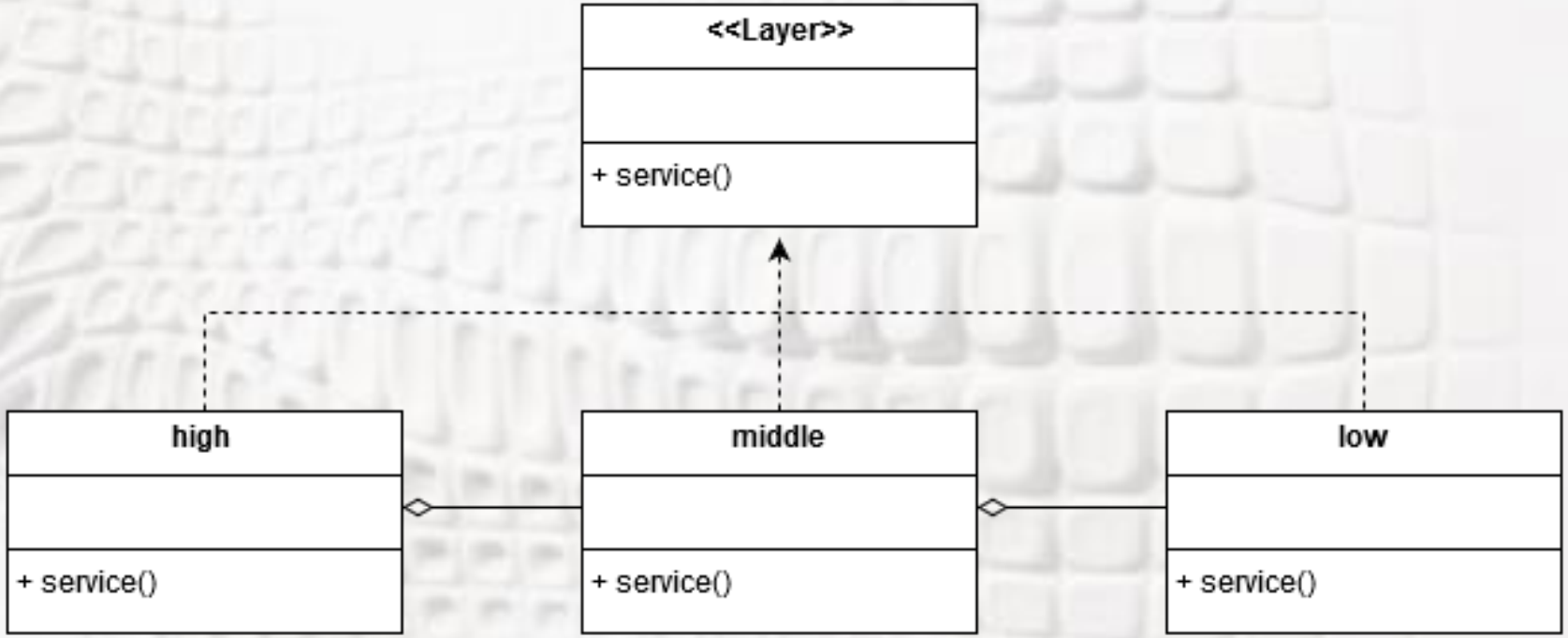
- широко приложими разделяне на функциите по принципа отгоре-надолу
- приложими са и при ОО имплементация
- проблем може да бъде достъпа до глобалните данни
- глобалните данни са модел на [разпределена] обща памет – затова са по-подходящи при мултипроцесорни машини или имитиращите ги платформи MPSV – и обикновено аргументите на обръщението са указатели, а не стойности

# Слоести архитектури

- групиране на различните нива в йерархията във функционално свързани слоеве от пакети класове, библиотеки от подпрограми (включени в т.нар. header files – заглавни файлове на проекта)
- интерфейсът на слоя се състои от интерфейсите на включените в него компоненти, а изпълняваната от тях функционалност – т.е. набора услуги – е протокола на слоя; интерфейсът му към нисколежащите слоеве се определя от техния интерфейс
- обаботката се декомпозира на заявки от по-висок слой към непосредствения по-нисък слой
- възможно е прескачане (“bridge”) в йерархията, но то е нетехнологично, тъй като изисква поддържането на пове4е от един интерфейс към слоеве с услуги; това се налага при необходимост от минимизиране на целия проект – напр. премахване на криптиращ слой
- протоколите на високите нива изпълняват приложно-ориентирани услуги, а на по-ниските – системно-ориентирани
- типично разслояване (6.10.1): потребителски интерфейс ↔ бизнес слой ↔ базови услуги ↔ услуги на ядрото
- клас диаграма на слоеста архитектура с имплементация на общ интерфейс от всички слоеве – 6.10.2

# Диаграми на разслоени архитектура – фиг. 6.10.1. и 6.10.2.

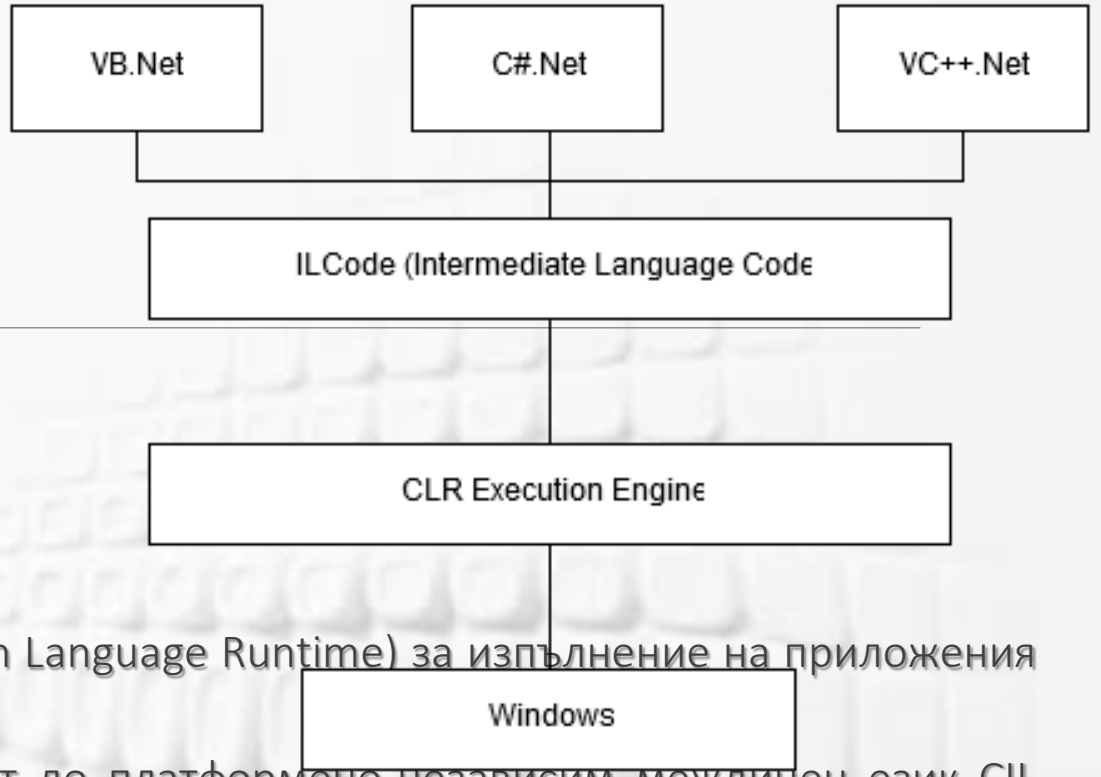


# Компонентно-базирано разслояване

- основен подход за капсулирането на услугите в слой е формирането на компонент, който се описва със своя интерфейс – напр .jar файл в JVM
- jar файлът (създава се с jar –cmf) представя всички класове от по-ниските слоеве и включва класовете от слоя, който имплементира
- компонентите  на отделните слоеве формират пакета на платформата – Java API
- всеки клас от jar компонента е достъпен за приложенията чрез своя интерфейс – стига да е включен в променливата на средата classpath

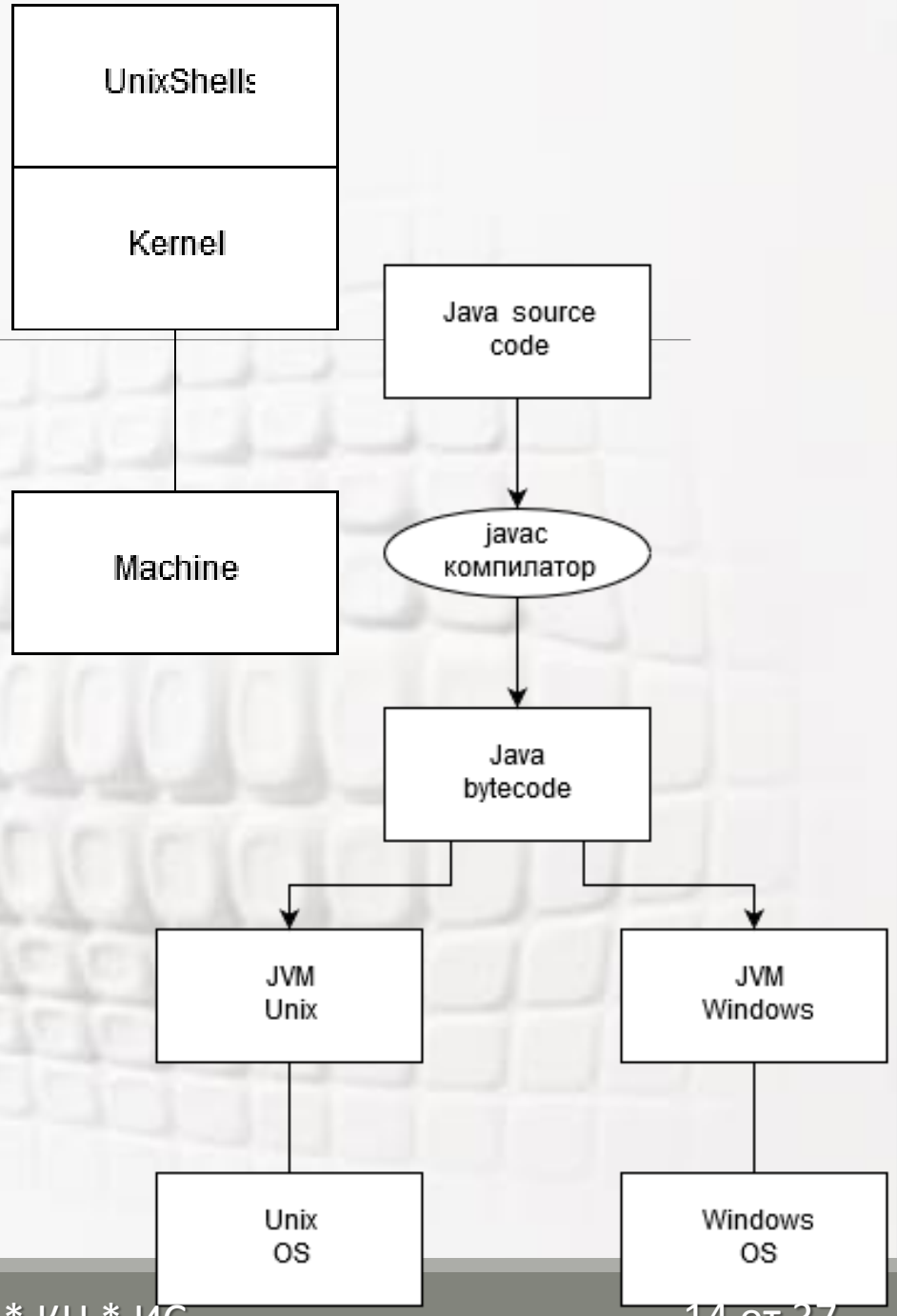
# Модели на разслояване




- OSI: App ↔ Pre ↔ Ses ↔ Tra ↔ Net ↔ DLL ↔ Phy
- Web-услуги: SOAP ↔ XML ↔ HTML ↔ TCP/IP
- Unix: shell ↔ core ↔ device drivers
- MS .Net: CLR ↔ JIT ↔CTS
  - .Net е технология, която осигурява платформата CLR (Common Language Runtime) за изпълнение на приложения на C#, VB.NET, C++/CLI аналогично на JVM – фиг. 6.13.
  - за прозрачност и преносимост приложенията се компилират до платформено-независим междинен език CIL (Common Intermediate Language),
  - по време на изпълнение CIL кодът (т. нар. „управляван код“) не се интерпретира като при други виртуални машини, а се компилира по начин, известен като JIT (Just In Time) компилация в платформено-зависим машинен код (native code) – за конкретната хардуерна платформа и операционна система
  - управлението на паметта, на нишките и процесите, защитата на паметта, верификацията и вътрешната компилация са системните услуги на CLR
  - CTS (Common Type System) дефинира всички базови типове данни и извършва конверсиите им. Тези типове са споделени между всички .NET езици и са стандартизирани в CLI.

# Виртуални машини

- виртуалните машини са слоест модел, който предоставя високо ниво на абстракция – програмен език или интерфейс за приложенията, при който скрива или обвива изпълнителната платформа
  - NB: ВМ представя основните абстрактни функции на системата като ги универсализира без да ги променя – напр. скрива интерфейсът към ОС – докато изпълнимите програми (C++) трябва статично да се прекомпилират за всяка ОС, както и за всеки тип процесор; понятието емулация (с което неправилно виртуализацията се смесва) означава изпълнение на функциите на дадена система от друга система (с принципно различни функции или организация) – напр. емулация на Unix върху MSDOS/Windows или емулация на PDA и Smart/Mobile Phones от настолен компютър
- Unix ВМ – фиг. 6.14.1
- MS .Net ВМ – фиг. 6.13
- JVM – фиг. 6.14.3



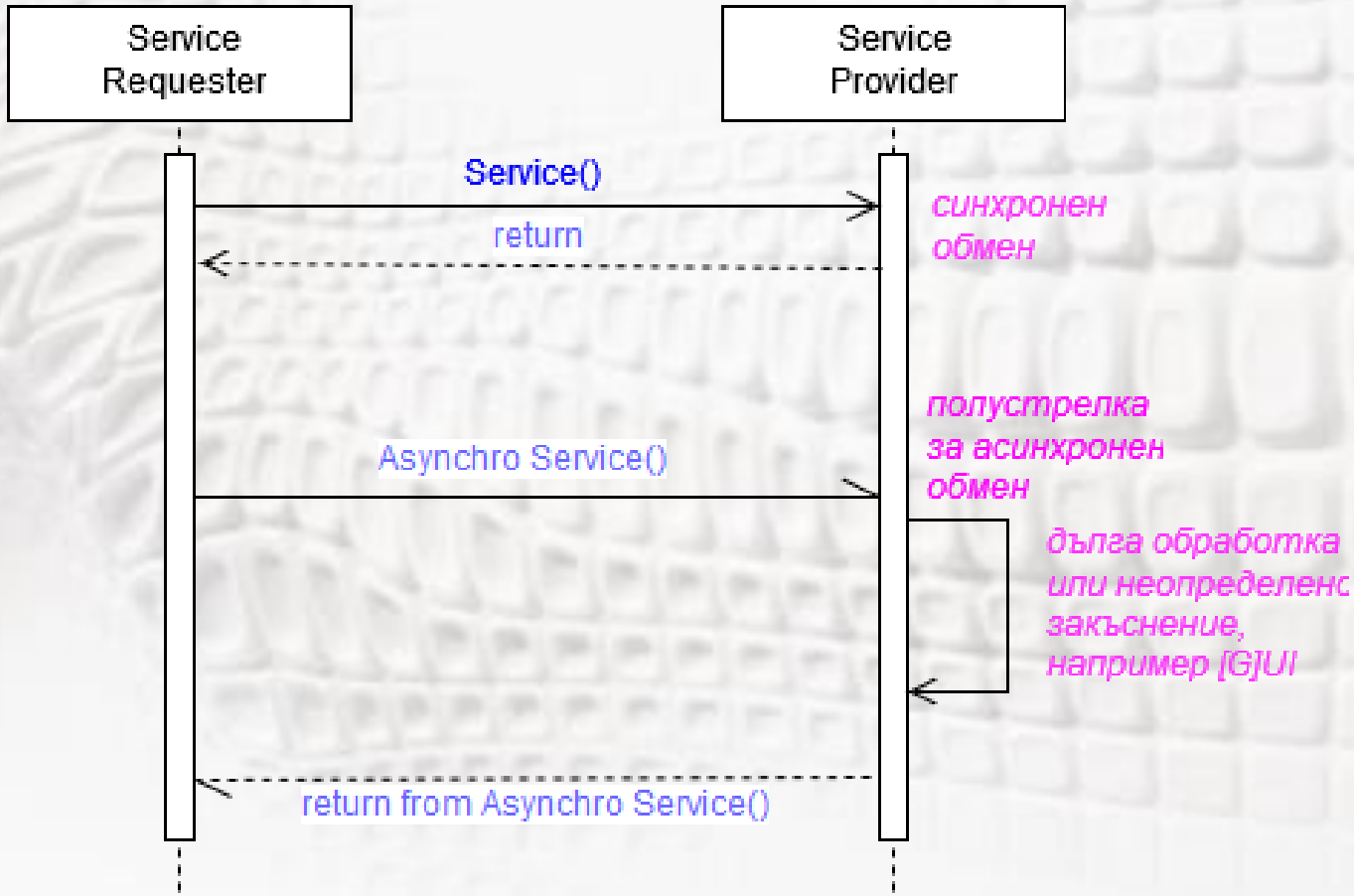
# Обхват на слоестите архитектури

- прилагат се за еволюционна развойна дейност, при която нивото на абстракция се повишава – принципа на проектиране е отдолу-нагоре, а не обратно
- всеки слой може да се разглежда като виртуална машина от определено ниво
- постига се
  - във високите слоеве – значителна прозрачност и преносимост на кода
  - в ниските слоеве – възможности за взаимстване на код (reuse) чрез промяна и добавяне на класове при запазен интерфейс на слоя
- подходяща за компонентни имплементации
- висок системен свръхтовар и по-ниска производителност – в сравнение с MS архитектурите
- свръхтоварът може да се преодолее с “мостове” през слоевете, но това намалява предимствата и смисъла на обща виртуализация
- слоевете имат тенденция да скриват настъпването на изключения от по-ниско ниво

# Асинхронни архитектури

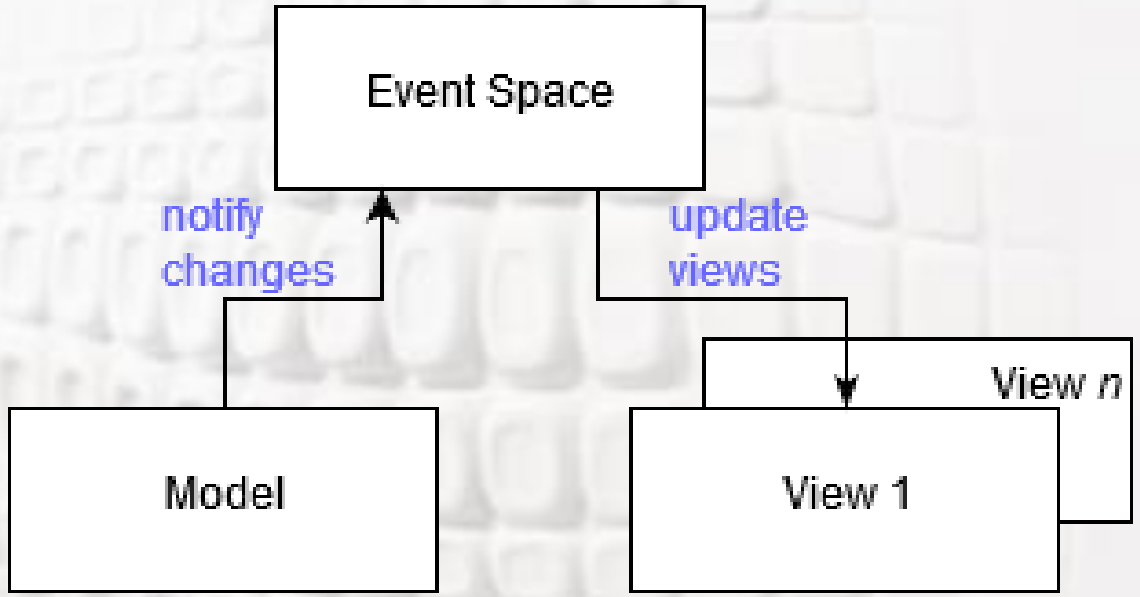
- базират се на неявни (implicit) асинхронни обръщения между обслужващите процеси
- асинхронният обмен може да бъде
  - свързан (online) – *без буфериране* – и двата процеса трябва да са активни, но не блокират изчакващо в точката на обмен – фиг. 6.16
  - независим (offline) – *с опосредяващ обмена процес-буфер* на съобщенията; приемащият процес може да не е активен в момента на изпращане на съобщението и обратно
- активният процес генерира съобщения, а пасивните процеси ги получават и евентуално изпълняват реакция
  - прилагат SW-шаблоните Производител/Консуматор (Producer/Consumer) или Издател/Абонат ≡ Наблюдател (Publisher/Subscriber, Observer)
  - управлението е по събитие (event driven) – където събитието е издаване на съобщение от издателя и получаване на съобщение от абоната
- в независимия вариант процесът-буфер алтернативно може да служи като
  - централизатор Message Topic на всички издадени съобщения и да ги препраща тематично до абонатите – един-към-много обмен
  - резервирана опашка Message Queue за един-към-един обмен

# Фиг. 6.16. – свързан (online) асинхронен обмен



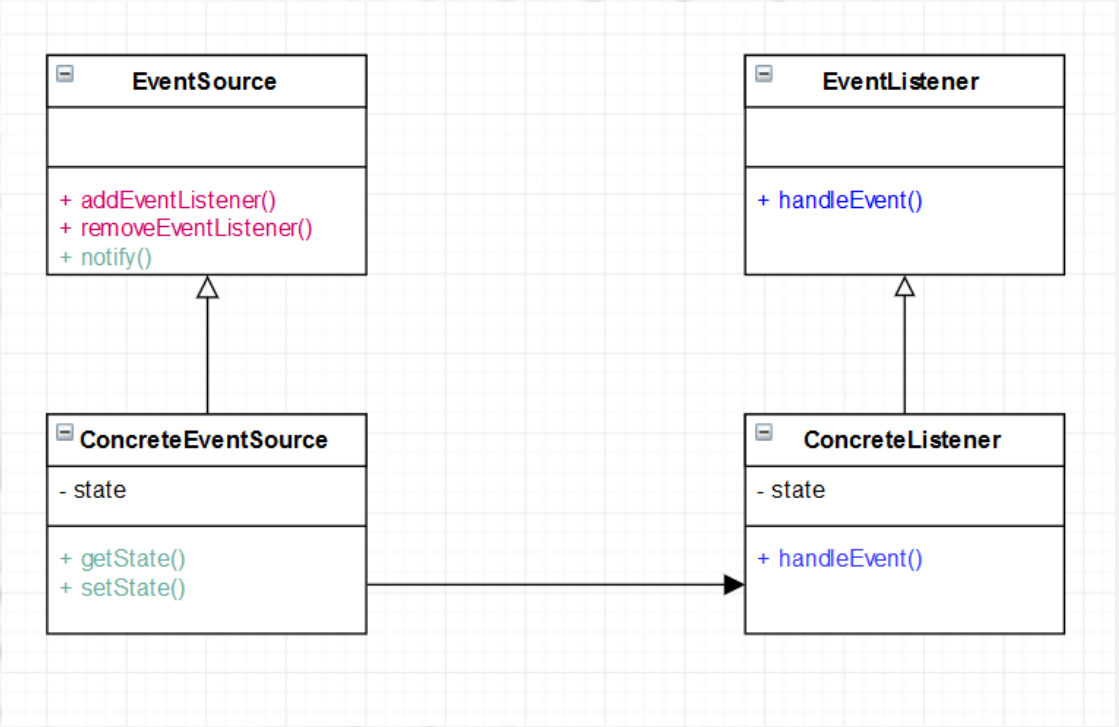
# Небуферирани асинхронни СА

- системата се декомпозира на 2+ части
  - генератори на събития (sources)
  - слушатели на събития (event listeners)
  - регистратори на събития, които опосредяват обмена и по-конкретно поддържат асинхронността и неявното (непряко) оповестяване на слушателите
- архитектурен модел на SmallTalk приложенията:
  - *n* пасивни графични компонента-слушатели $View_n$ се регистрират в активно (т.е. инициативно) пространство на събития EventSpace за съобщения от даден генератор на събития Model – фиг. 6.18.



# Небуферирани асинхронни СА – SmallTalk фиг. 6.19.

- клас диаграма на архитектурата – фиг. 6.19.
  - класът Event Source осигурява операции за регистриране на слушател и за уведомление за събитие
  - класът Event Listener осигурява операция за анализ на събитието и генериране на реакция
- подходящ модел за приложения с GUI и слабо-свързана логика, чиито модули се представят с машина на състоянията и имат недетерминистично поведение (поради което по-сложна настройка и тестване)
- налична е значителна поддръжка от междинни компоненти
- елемент на синхронност (👎) е началната регистрация и желателна де-регистрация – средство за актуализиране на списъка активни процеси
- сравнително ниска производителност и голям системен свръхтовар



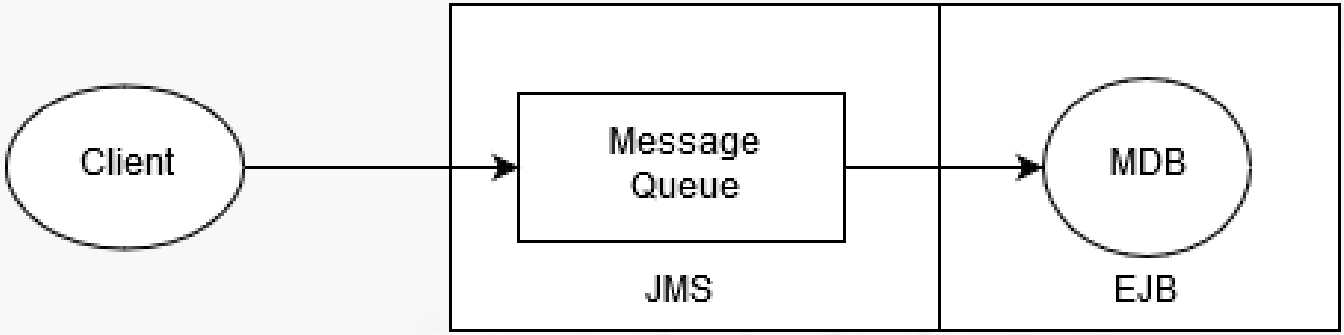
# Буферирани асинхронни СА

- системата е
  - контекстна (data-centric),
  - слабо свързана (не се чака потвърждение за получаването на съобщенията и обикновено не се получава отговор след обработката) – но с надежден обмен
  - декомпозира се на 3 части
    - генератори на съобщения (producers)
    - консуматори на съобщения
    - услуга за асинхронен буфериран обмен на съобщения – MOM (Message Oriented Middleware
- висока скалируемост, надеждност, p2p и CS приложения
- за системна поддръжка (мрежи, телекомуникации), бизнес приложения (бюлетини – новини, метеорология, групи по интереси; транзактивно банкиране и е-търговия)
- поддържа опашки (Message Queuing, MQ) и тематичен обмен (Message Topic, Publish/Subscribe Messaging P&S)
- атрибути на съобщенията са ID, заглавие (header) и тяло
- клиентите на системата обменят съобщения инициативно или пасивно, като адресацията е на базата на идентификатор, получен при началната регистрация на клиента в услугата за обмен

# MOM

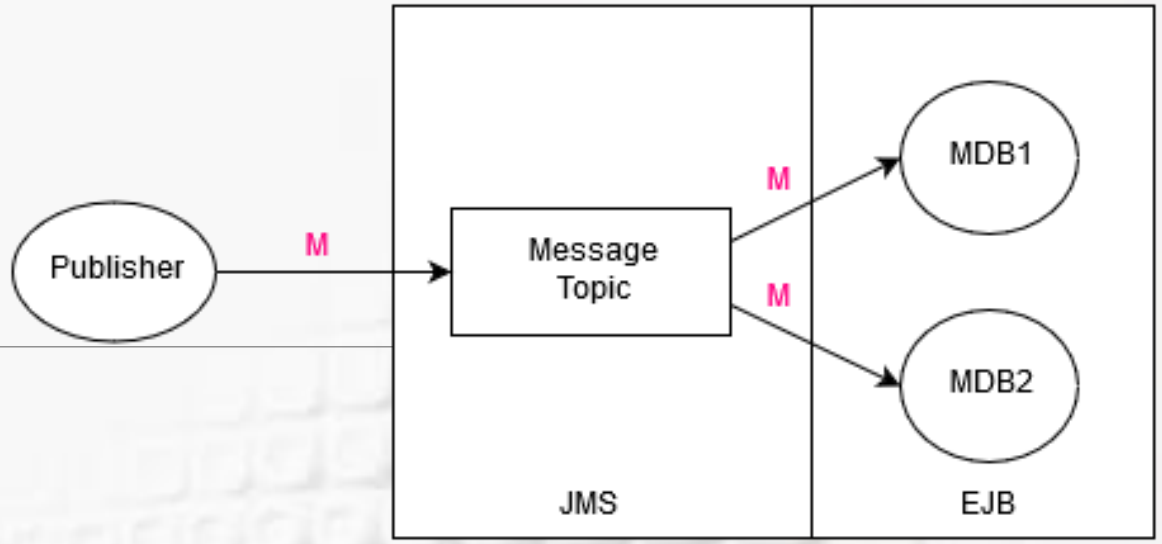
- MS MQ
  - (http://en.wikipedia.org/wiki/Microsoft_Message_Queuing)
- IBM WebSphere MQ (бивш MQseries)
  - (http://en.wikipedia.org/wiki/WebSphere_MQ)
- JBossMQ (Java Message Server)
  - (http://www.jboss.org/community/docs/DOC-10525;
  - http://java.sun.com/products/jms/tutorial/1_3_1-fcs/doc/jms_tutorialTOC.html)
- Oracle [бивш BEA] WebLogic JMS
  - (http://e-docs.bea.com/wls/docs92/index.html)

# p2p (point-to-point) обмен




- обменът е 1:1 – всяко съобщение има точно 1 получател
- елементи: изпращач на съобщения, получател и асоциирана с получателя опашка, която поддържа асинхронността на обмена
- съобщенията до даден клиент-консуматор се съхраняват в неговата опашка-буфер до извличането им или до изтичането на срока им
- пример – блокова диаграма на p2p обмен в EJB (Enterprise Java Beans – Java компонентна библиотека за бизнес приложения) – фиг. 6.22
- получател е MDB (Message Driven Bean)
  - изпращач е клиентски процес
  - опашката може да се организира чрез JMS (Java Message Service http://java.sun.com/products/jms/) API – системно приложение за поддържане на универсален асинхронен обмен

# Pub/Sub (P&S) обмен

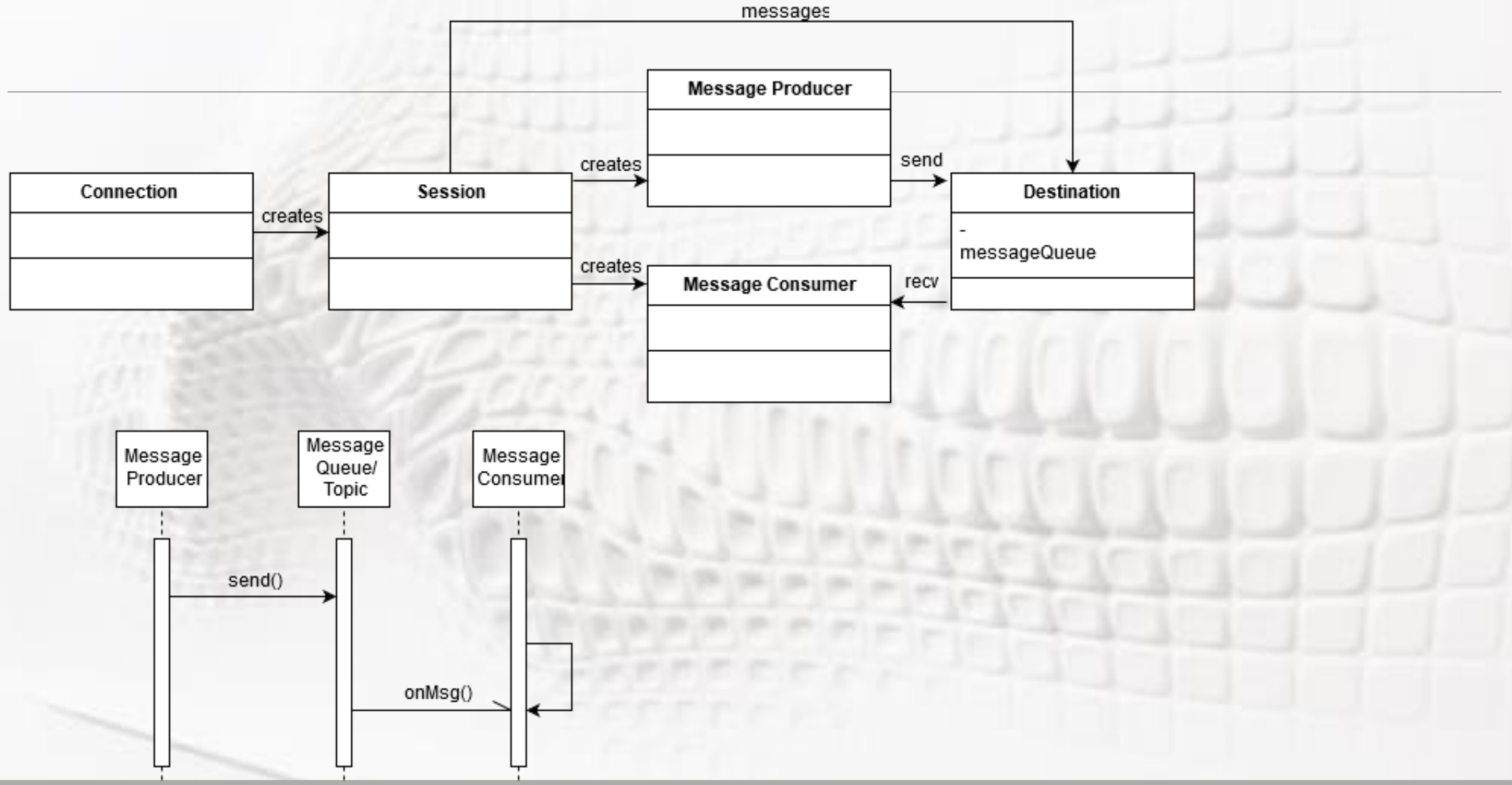



- тази СА се базира на централизатор (hub), поддържащ асинхронния и непряк обмен на съобщения между издатели и абонати по теми (topics) – тип бюлетин
- инициативата в обмена принадлежи на източника на съобщението – на издателя спрямо бюлетина и на бюлетина спрямо абоната – така се постига максимална асинхронност
- вариант е устойчивия абонамент (durable subscription), при който абоната получава и съобщенията по дадена тема, издадени преди неговата регистрация в бюлетина
- блок-диаграма на P&S СА фиг. 6.23 – системата се базира също на JMS MDB/EJB, но за разлика от p2p при P&S крайните получатели на дадено съобщение могат да бъдат повече от един – всички регистрирани (и евентуално бъдещите!) абонати по темата (или темите), за които е издадено съобщението
- при разгърната P&S СА клиентите – издатели и абонати – са отдалечени разпределени процеси без никаква явна връзка помежду си, като абонатите обикновено изпълняват информационни услуги за трети клиенти – напр. сесии със СУБД

# JMS комбинирана (p2p + P&S) СА

- клас-диаграма и д-ма на последователността – фиг. 6.24
- по отношение на услугата на обмена клиентите (производители и консуматори на съобщения)
  - се регистрират
  - откриват сесия за изпращане или приемане на съобщения
  - създават опашка или тема
- JMS (и др. MOM) поддържа следните контроли за надеждност и QoS на обмена
  - обмен с потвърждение от опашката/бюлетина
  - означаване на съобщението като обмен без загуба
  - установяване на приоритет на съобщенията
  - срок на съобщението (expiration)

# JMS комбинирана (p2p + P&S) СА – фиг. 6.24.



# Обхват на асинхронните СА

- 👍 подходящи са за слабосвързани системи с устойчив неявен обмен на съобщения, при които обменящите процеси са а н о н и м н и – не знаят идентичността на комплементарния процес/и (в т.ч. и неговия интерфейс!)
  - 👍 т.е. времева и локационна независимост
- 👍 висока скалируемост и заменимост на компонентите
- 👍 подходящ за динамично настройваеми разпределени изчисления (при асинхронен алгоритъм!)
- 👍 подходящи СА за пакетна обработка
- 👍 подходящи за интегриране на наследени приложения (legacy systems) в съвременни проекти
- 👎 независимостта между обменящите клиенти ограничава логиката на приложенията:
  - 👎 логиката на клиентите трябва да е независима от получаването (и неполучаването) на конкретни съобщения
  - 👎 не се идентифицира източника и няма пряк обмен с него
- 👎 усложнена логика на клиентите поради изискването за гъвкавост т.е. всеки клиент се самоконтролира (контраст с йерархичните и централизираните системи)
- 👎 възможност за тясно място (bottleneck) – по време (производителност на опашката/бюлетина) и по пространство (размер на опашката/бюлетина)

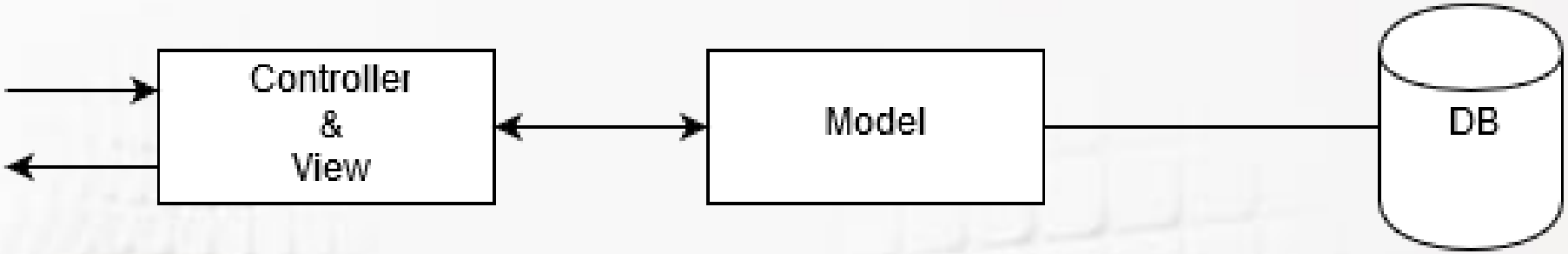
# Интерактивни софтуерни архитектури

- поддържат интензивен потребителски интерфейс
- за целта декомпозицията на системата е на 3 функционални модула
  - модул за представяне (изглед) – с потребителски интерфейс – за представяне (в т.ч. графично или мултимедийно) на изходните данни и също намеса на потребителите в обработката (т.е. вход за данни и контрол)
  - модул данни – поддържане на данните с базова функционалност върху тях
  - модул за управление – системни комуникации, управление на процесите, инициализиране и конфигуриране на модули данни, управление на изгледи
- поддържа множество (и то адаптивни) изгледи за даден набор данни
- слабо свързана архитектура, която поддържа явни и също неявни обръщения към метод – респ. RMI и модел регистрация/уведомление (notification)
- две категории ИСА: PAC (Presentation-Abstraction-Control) и MVC (Model-View-Controller)
  - аналогията е P-V, A-M и C-C
  - прилагат различно управление:
    - PAC е с йерархично (разслоено) и разпределно управление, при което системата се формира от набор кооперирващи агенти на три нива – базово ниво агенти на общи данни и бизнес логика, ниво на изгледите за локални данни и средно ниво агенти координатори на изгледите; всеки агент интегрира P, A и C компоненти
    - в MVC агентите са равнопоставени

# MVC

- основен модел за сървърни приложения с Web-клиенти за достъп – е-бизнес, е-управление, системи за потребителски профили и т.н.
- специализация: промени в контекста (данните) се представят динамично т.е. в реално време при отдалечени клиенти
  - изгледите се базират на интуитивни графични интерфейси с приложение на контекстно настройваеми “кожи” и фокусиране на интерфейса – етикети, бутони, изборни полета и др. компоненти от тип widget (в-ж л-я 9.)
- приложна компонентна платформа за проектиране на MVC е напр. Java Swing (http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/package-summary.html)
- трите дяла на MVC имат следната специализация:
  - контролерът регистрира, подрежда и предава последователността от потребителски заявки; настройва изгледа вкл. динамично и управлява останалите модули ва СА – стартиране, настройка, обмен
  - моделът изпълнява базовите функционални услуги, като капсулира контекста (непрозрачна обвивка на данните); при СА MVC I той не поддържа пряк интерфейс с присъединените към него изгледи
  - изгледът е динамично настройваемо графично представяне на заявена част от контекста

# MVC I

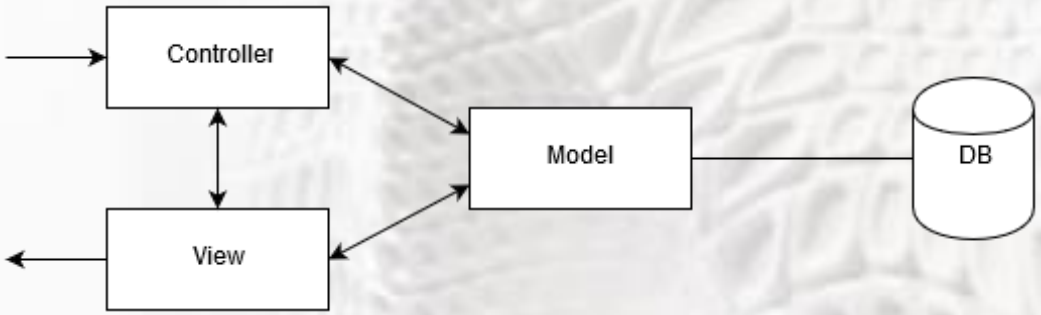


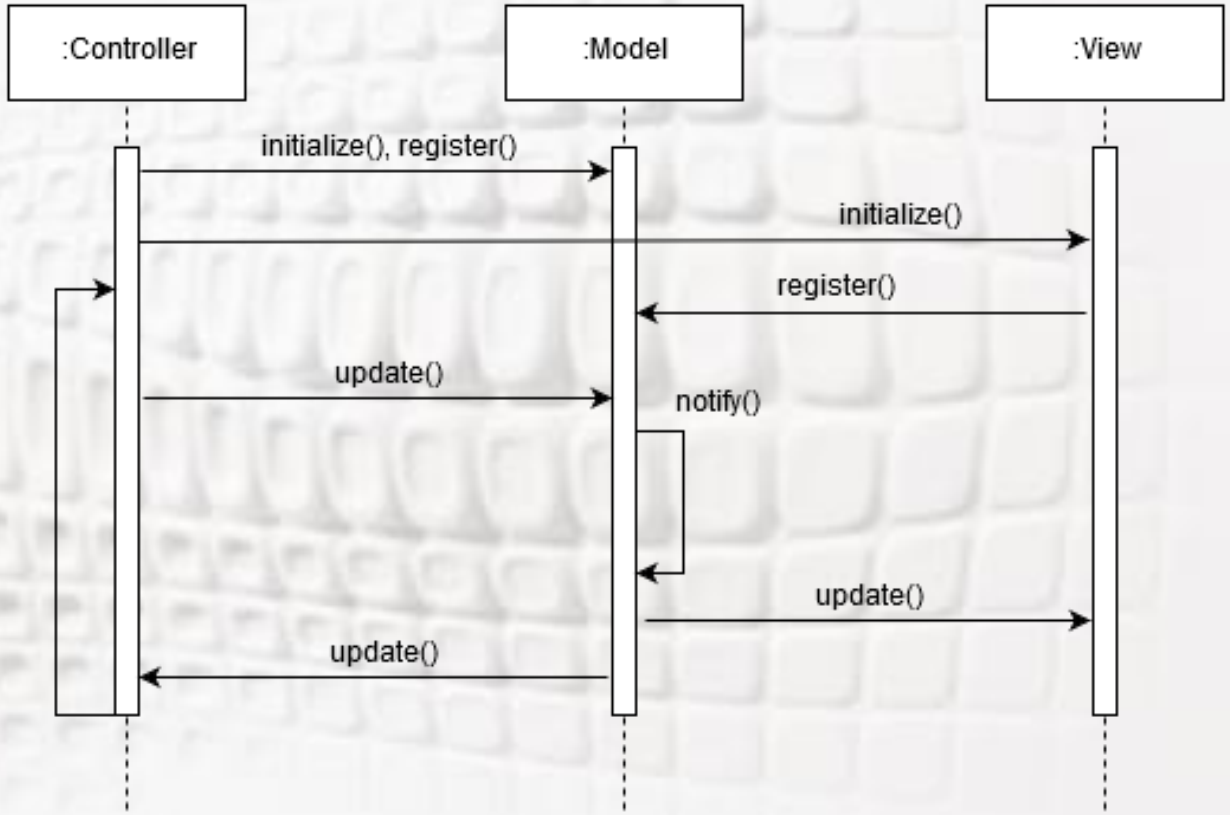

- компактна и базова двуделна имплементация на MVC, при която контролерът и изгледът са интегрирани в един модул C/V
- C/V се регистрира и присъединява към даден модел като се абонира за уведомления за промени в контекста, които представя в реално време, и служи като Вх/Изх на модула за данни – фиг. 6.29
- C/V
  - поддържа форми за потребителски вход – текстови полета, радиобутони (за алтернативен или множествен избор на опции) и др. уиджети
  - при промяна във входните данни ги валидира и генерира заявка към модела с новото съдържание
  - представя резултата, който се генерира от модела (на базата на заложената в модела функционалност)
  - представя промените в контекста на модела, за които е абониран (без заявка от потребителя) – модел “активен контекст”
- MVC I е приложим за по-прости приложения с компактен GUI

# MVC II

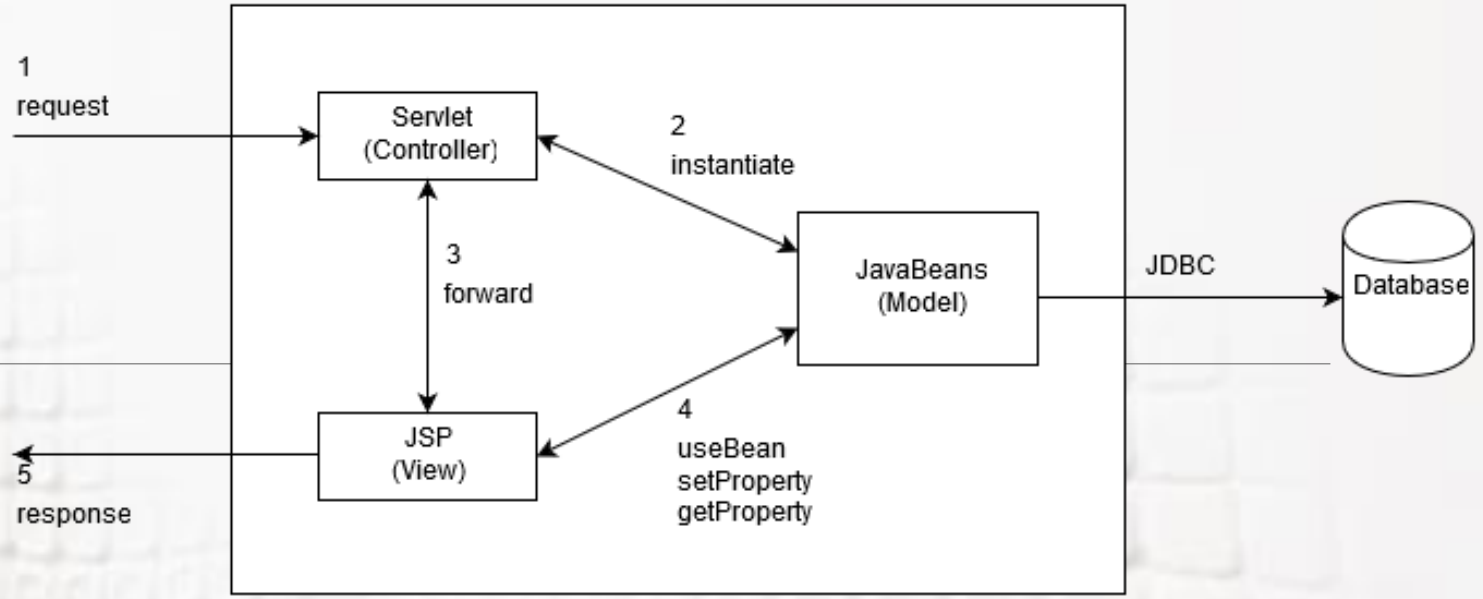
- при MVC II контролерът и изгледът са самостоятелни, а евентуално и отдалечени процеси
- допълнителна функция на контролера е да инициализира връзката между изгледа и модела и управлява обмена между тях
- контролерът и изгледът се регистрират в модела и се уведомяват разпределено за промените в контекста
- разделянето позволява самостоятелна имплементация и технологии за V и C
  - това способства за проектиране на сложна функционалност и също за самостоятелна еволюция на двата модула – по-специално на изгледите, които се поддържат от бързоразвиващите се графични технологии
- блок-диаграма, клас-диаграма и последователностна диаграма на MVC II – фиг. 6.30
  - инстанциите на класовете V и C са “сдвоени”, като множество двойки се поддържат от един модел
  - класът модел агрегира колекция от класове с различни функции върху базата данни

# MVC II: блок-диаграма, клас-диаграма и последователностна диаграма – фиг. 6.30



# MVC II с Java

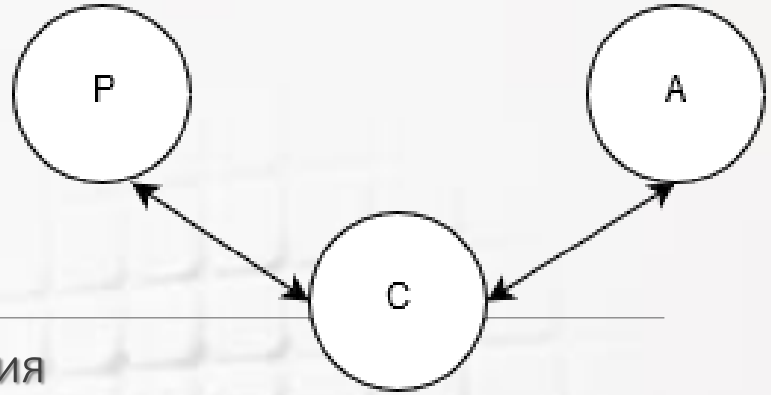



- блок-диаграма на MVC II СА, базирана на Java технологии – фиг. 6.32
  - JSP (Java Server Pages) се използва за V; EJB (Enterprise Java Beans) + JDBC (Java Data Base Connectivity) се прилагат за развитие на M; C може да се имплементира като произволно сървърно приложение – напр. с Java Servlet технологията (сървлетите са Java сървърни приложения без потребителски интерфейс, които се инициализират от резидентни сървърни програми напр. Tomcat – подобно на аплетите, които обаче се изпълняват в клиентската част от браузъра)
  - контролерът получава потребителска заявка от графичен или текстов интерфейс (1), стартира необходимата инстанция на модела (2), селектира и стартира необходимия изглед (3) – с което управлението се предава към изгледа
  - изгледът получава данни от модела (4) и ги представя графично (5)
  - разгледайте аналогичните технологии в платформата MS .Net – ASP/ADO

# Обхват на MVC

- това е базовата архитектура за приложения с интензивен потребителски В/И с динамично представян на данните и с възможност за самастятелна имплементация на модулите
- поддържа се от множество професионални платформи за шаблонно развитие на приложенията
- не поддържа агентно-базиран информационнен обмен, характерен за системите с редуциран потребителски интерфейс – автономни и вградени системи, роботи, автонавигатори и др.

# PAC

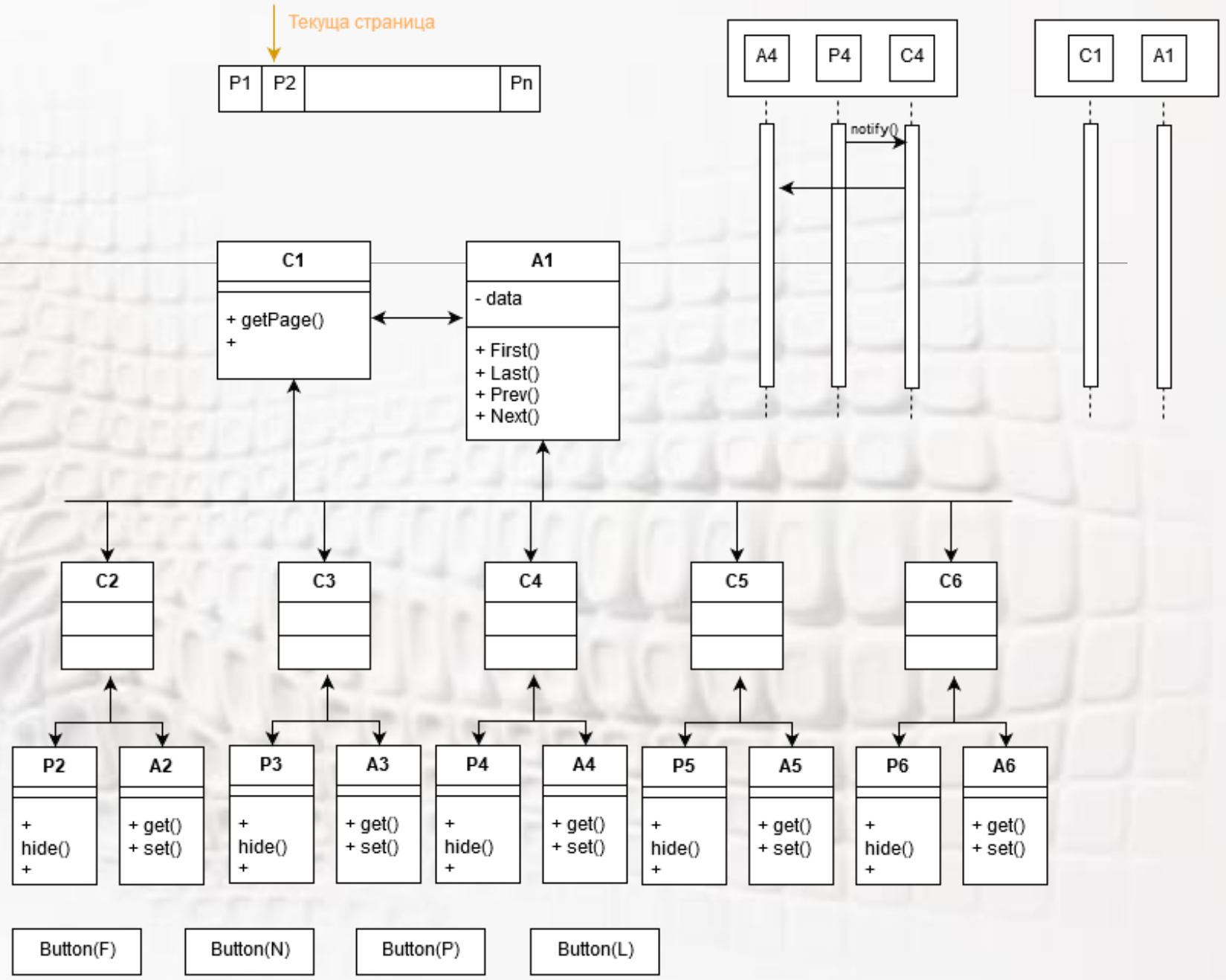



- PAC е развитие на MVC, което поддържа агентен обмен на съобщения
- системата се състои от множество специализирани (т.е. с различни функции) агенти, декомпозирани на трите модула – P, A и C;
- декомпозицията на даден агент разделя неговия потребителски интерфейс (P) от функционалността, която поддържа (A) и от модула му за обмен с др. агенти (C) – фиг. 6.34
  - презентационния модул на агента е опция (съществуват агенти-посредници без потребителски интерфейс)
  - контролния модул е задължителен, освен комуникациите с отдалечени агенти, той управлява достъпа до функциите на агента – P и A са слабосвързани процеси без пряк обмен
  - абстрактният модул капсулира данните и операциите на агента

# PAC-приложение

- примерно PAC-приложение (клас- и последователностна диаграма фиг. 6.35) за преглед на отдалечен страницира документ
  - с 4 бутона – за първа, предишна, следваща и последна страница – поддържани от агентите Æ2 ÷ Æ5 съответно
  - Æ6 – за графична интерпретация на страниците от документа по съответен стандарт
  - Æ1 е агента за достъп до документа в БД
  - C1 приема заявките от Ci (i = 2 ÷ 5), настройва A1 на съответната страница, приема я от него и я предава на заявителя – Æ1 няма нужда от P1
  - C1 съобщава на Ci за настройки на бутоните от Pi (напр. избледняване на бутони “следваща стр.” и “последна стр.” ако прегледа достигне последната страница) и предава на C6 съдържанието, което се представя от P6
  - Ai поддържат контекста на съответните агенти – напр. предпочитан изглед на бутон, текущото му състояние
  - A6 поддържа контекста на представяната страница – напр. декодиращ метод, кеширани страници

# PAC-приложение – фиг. 6.35.



# Обхват на PAC

- прилагат се за интерактивни системи от кооперирaщи специализирани информационни агенти
- слабосвързана разпределена система – комуникациите са неблокиращи асинхронни
- 👍 добри възможности за заменимост, еволюция на агентите, ескалиране на системата
- 👍 поддържа еднакво многонишкови и многопроцесни разпределени приложения
- 👎 значителен свръхтовар особено при групови комуникации
- 👎 непряк (бавен) обмен между контекста и представянето му
- 👎 имплементацията на A и P е зависима от тази на C – затруднение при проектирането
- 👎 усложнени операции за откриване на броя и идентифициране на текущите агенти