

5. Обектни, потокови и контекстни модели на софтуерната архитектура

ВАСИЛ ГЕОРГИЕВ

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

 v.georgiev@fmi.uni-sofia.bg

Съдържание

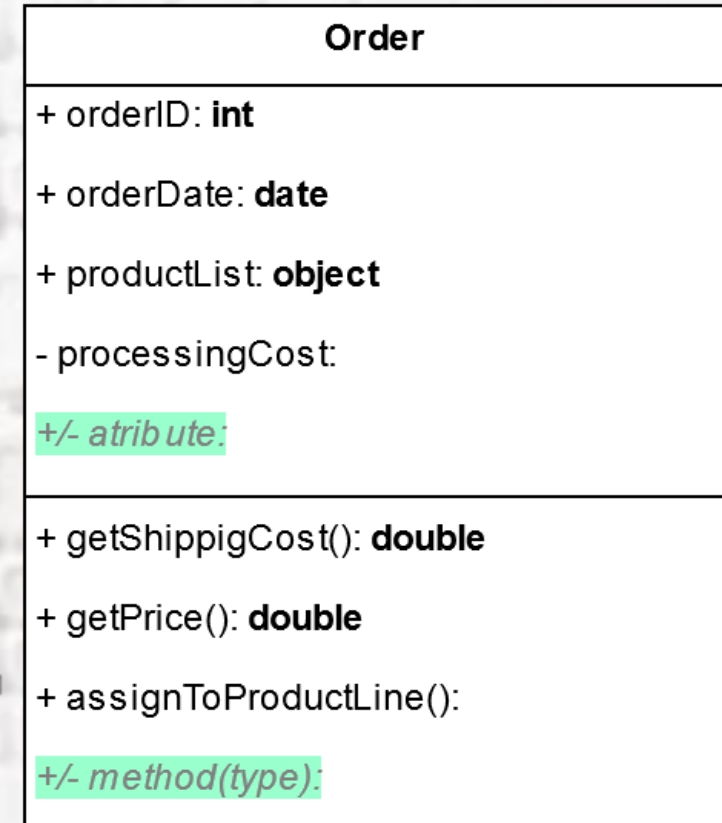
- ОО-архитектури
 - абстракции, структури, отношения
 - анализ и принципи на проектиране
- Поточкови архитектури
 - пакетна обработка
 - архитектура с канали и филтри
 - контролна архитектура
- Контекстни архитектури
 - с хранилища
 - тип Черна дъска

ОО-принципи

- развитие на езиковите принципи при усложнена софтуерна архитектура – първоначално за симулационни модели (Simula67, C++); Интернет приложения (Java, C#); към компонентно базирани технологии
- прилагат се три основни принципа
 - капсулиране – видимост на функциите и прозрачност за имплементацията . Например скрит вътрешен контекст и процедури – частни променливи в класовете, неустойчиви; публичен интерфейс – устойчив
 - наследственост – адаптивност на кода чрез наследяване и допълване на спецификациите – т.е. от общо (родителски клас) към частно (наследен клас, дериват)
 - полиморфизъм – адаптивна функционалност чрез развитие на наследяването
 - отмяна и предефиниране на атрибути в дериватите (вертикален полиморфизъм) или
 - презареждане на нов контекст за същия клас – хоризонтален полиморфизъм

ОО-софтуерна архитектура

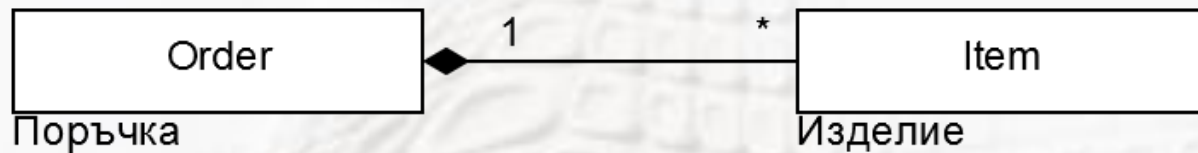
- Абстрактни типове данни
 - капсулиране на данните с функциите върху тях
 - стандартни библиотеки от типове
 - публични и частни атрибути на типовете
- класовете са имплементации на абстрактните типове данни с публичен интерфейс от атрибути и операции
- обектите са имплементации на класове, които се явяват техни «типове»
- **Фиг. 5.4:** UML-спецификация на клас с +/- модификатори на достъпността на атрибутите и операциите –
- Статични отношения между класовете:
 - конструкцията на комплексни класове от класове
 - композиция
 - наследяване
 - статична консистентност (т.е. логичност) на зависимите класове – като при базите данни
 - агрегация,
 - асоциация
- Динамични отношения между класовете – обмяна на съобщения



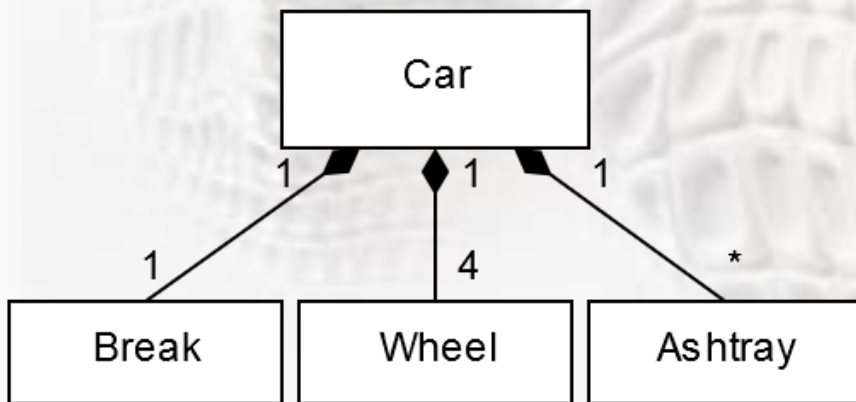
Композиция, агрегация и асоциация

- **композицията** е дефинирана на клас като съставен от други класове
 - компонентите са активни докато и активен съставния клас и не се включват в други класове (пресилено ограничение за garbage collection – чрез конструкторите и деструктурите на класовете)
 - в UML – плътен ромб към главния клас с етикети на мощността – [5.5.1](#)
- **агрегацията** е аналогично отношение на класовете, но без изброените ограничения – [5.5.2](#)
- **асоциацията** е обобщена композиция – [5.5.3](#); характеризира се с
 - име (етикет), което отразява свързващата функционална логика – напр. «Customer places an/[some](#) Orders»
 - мощностите на асоцииране
 - 2 асоциативни типа на връзката между двата класа – задават тип композиция към инициращия клас
 - навигационната посока към иницирания клас – т.е. указателите на асоциираните класове са налични като атрибути в инициращия клас (плътна линия)
 - зависимост посока към зависимия клас – зависимия клас извиква операция на асоциирания клас или променя негов атрибут (пунктир)
 - инициращият клас може да асоциира повече от един класове

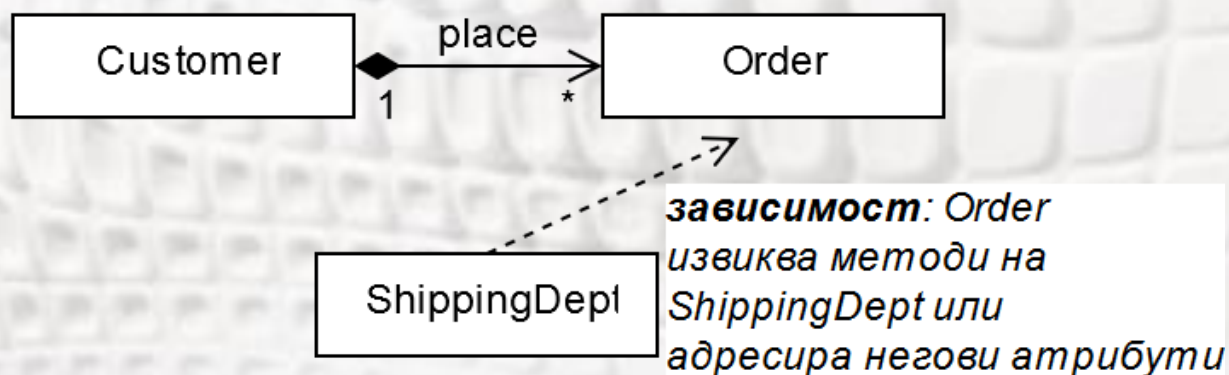
Композиция, агрегация и асоциация – фиг. 5.5.*



Фиг. 5.5.1 Композиция (**HAS-A**)



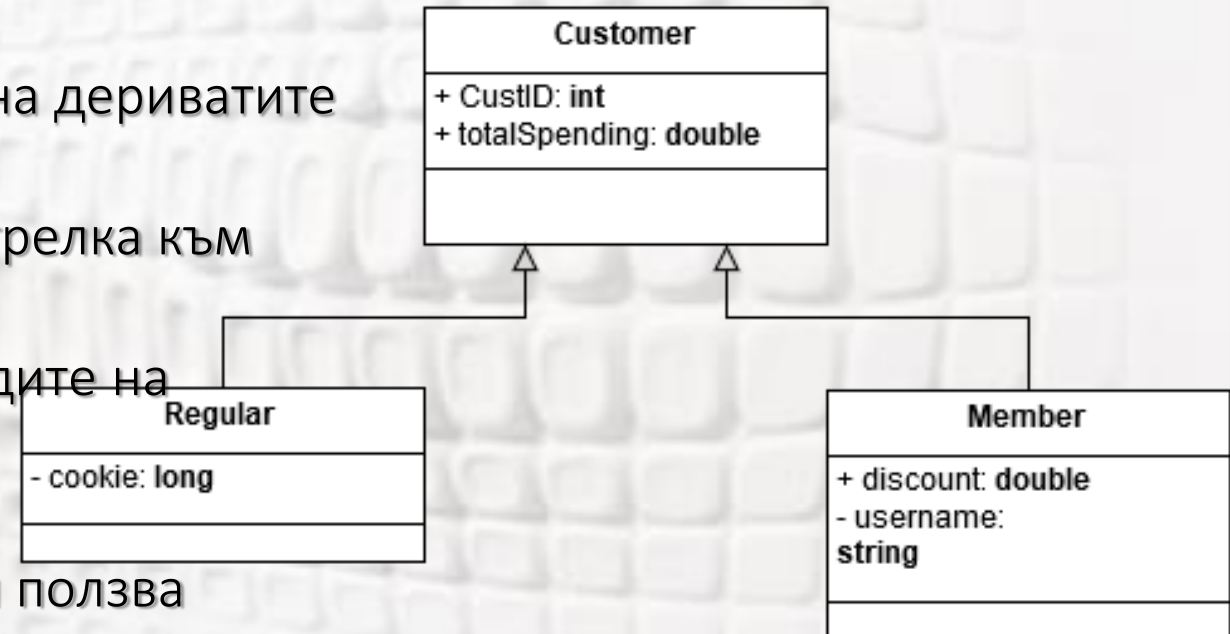
Фиг. 5.5.2 Агрегация (**HAS-A**)



Фиг. 5.5.3 Асоциация (**USES-A**)

Наследяване и полиморфизъм

- наследяването отразява взаимодействие на повтарящите се атрибути – деривата наследява всички публични атрибути (без частните – възможни изключения)
- полиморфизмът е механизъм за диверсификация на дериватите при изпълнение – 5.7
- в UML наследяването се означава с триъгълна стрелка към основния клас
- в примера двата деривата се различават по методите на идентификация:
 - клиентът зарежда cookie в брауъра си
 - регистрираният потребител изпраща парола и ползва отстъпка
 - и двете функционалности отсъстват в базовия клас



Фиг. 5.6 Наследяване

Обхват на наследяването и композицията

- и двете черти поддържат заимстването на атрибути между класовете (reuse), но с различен обхват на приложение съгласно принципите:
 - наследяване се прилага при is-a отношение между деривата и базовия клас
 - композиция (или агрегация) се прилага когато отношението е has-a
 - пример: базови класове Person и University, класът Student може да бъде дериват на двата класа или да има атрибути с указатели към двата класа или комбинация от двата подхода
 - Student IS-A Person → Student е уместно да бъде наследствен дериват на Person
 - Student HAS-A University → Student е уместно да има атрибут с указател към University
 - наследяването е противоположно за капсулацията (локалността) на кода тъй като промяна на атрибут в базовия клас предизвиква каскадни промени в дериватите -
 - пример (5.8) - Student и Professor като деривати на Person (легитимно, но ниска капсулация) и като агрегирани PersonalHandler (с прозрачна конверсия на обръщението към атрибути)

Фиг. 5.8.: наследяване "is-a" и агрегиране "has-a" (Student/Professor агрегират Handler).

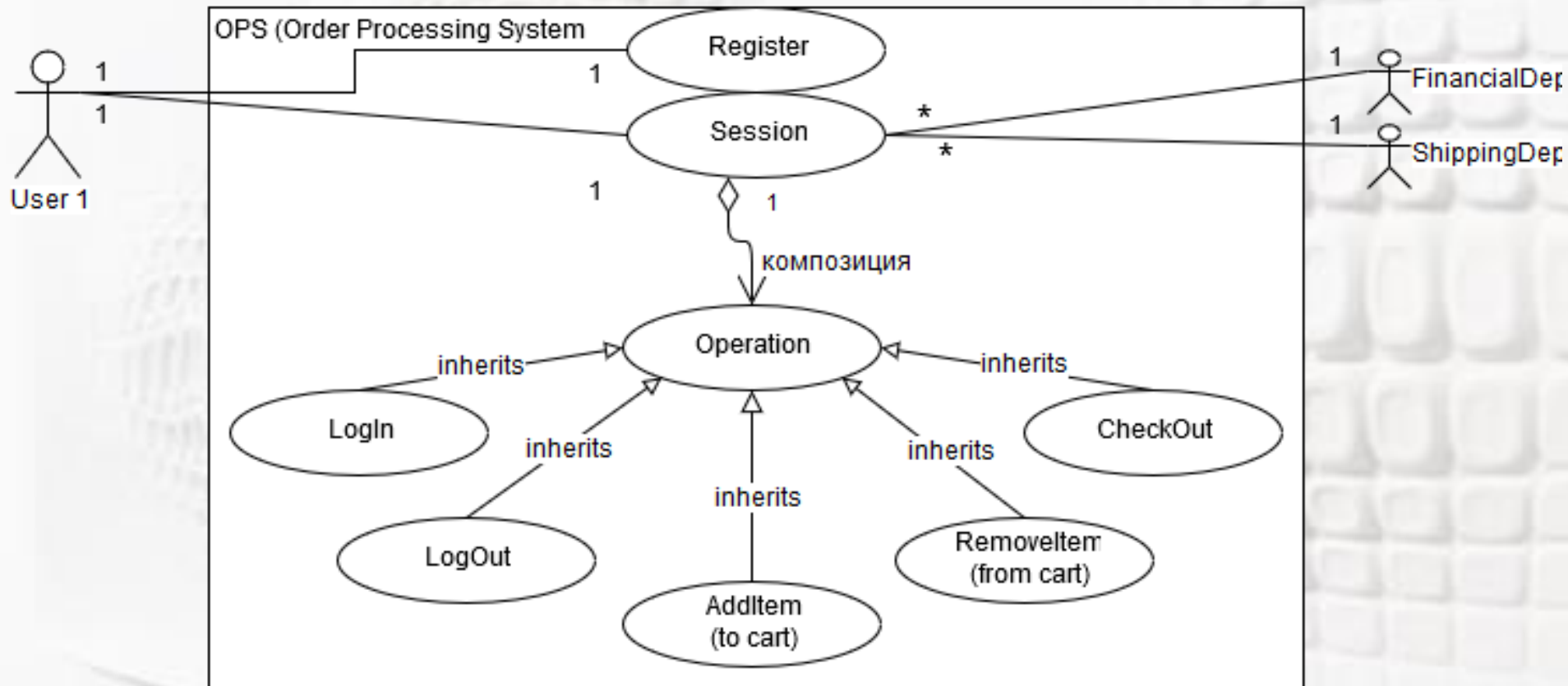


Фиг. 5.7 Наследяване "is-a" и агрегиране "has-a"

ОО-анализ – диаграма на случаите на употреба

- анализът предхожда проектирането и имплементацията и се състои в структуриране на предметната област и представянето ѝ като набор класове с определена функционалност
- обикновено се състои в описание на потребителския сценарий чрез диаграма на случаите, от която се извлича и аналитичната (или принципна) клас-диаграма
- диаграма на случаите (Ivar Jacobson 1987) – пример за OPS (Order Processing System) **5.10**:
 - определя типовете потребители на системата – напр. клиент, счетоводство, доставка
 - определят се основните случаи, които ще се детайлизират като [една или повече] операции в етапа на проектирането – напр. случая добавяне на изделие в пазарската количка би изисквал и операция със складовата БД

Диаграма на случаите на употреба на OPS (фиг. 5.10.)

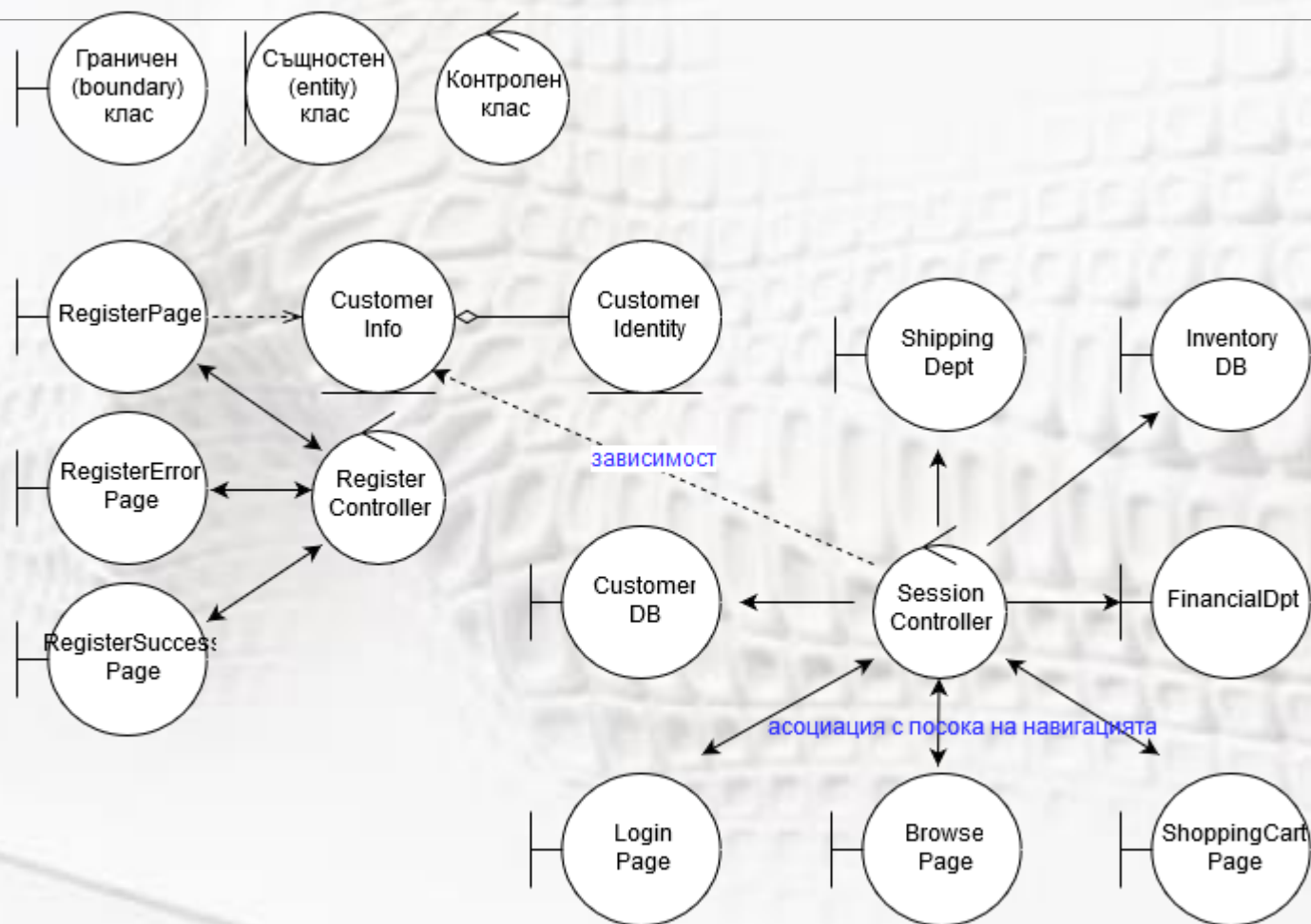


Фиг. 5.8 UseCase Diagram на OPS

ОО-анализ – принципна клас-диаграма

- принципната КД е абстрактно описание на класовете на системата – по-близко до сценариите и функционалността, отколкото до имплементацията (не отчита производителност на модулите, технологии и технологичност на проектирането и експлоатацията)
- състои се от гранични, същностни и контролни класове (boundary, entity, control)
 - граничните класове се извличат от интерфейсите случаи и са ориентирани към имплементация
 - с GUI (Web форми, прозорци, браузър-плагини) или
 - като междинни интерфейси (middleware wrappers) към други системи
 - същностните класове отразяват информационния слой – напр. клиентската или пордуктова идентичност са същностни класове
 - контролните класове отразяват отделните случаи т.е. операциите, които свързват граничните и същностните класове
- пример – 5.12 – принципна КД на OPS

Фиг. 5.12 – принципна клас-диаграма на OPS



ОО-проектиране

- проектирането е самостоятелна фаза в развойната дейност на разпределените система
 - ☞ може да се приложи подход, различен от този на фазата на анализа – потоков (event driven), контекстен (data driven), структурен (с функции)
- целта му е декомпозиция на системата на технологични модули – при ОО – класове
 - класовете се описват с техния интерфейс т.е. публичните им атрибути и операции, и се специфицират след това на фазата на имплементацията
- различават се високо и ниско ниво на проектирането
- високото ниво идентифицира класовете напр. с приложение на CRC-карти и клас-диаграми за статичните отношения (specification/compile time) между класовете
- ниското ниво детайлизира проектираните класове и тяхното динамично взаимодействие (run time) с диаграми за взаимодействието (най-често с диаграми на последователността или на комуникациите) и на машината на състоянията (state machine) – като се използват диаграмите на случаите от фазата на анализ

ОО-проектиране – стъпка 1.

- прилагат се CRC карти (Class-Responsibility-Collaborator – Kent Beck & Ward Cunningham, 1989) и/или клас диаграми за пълно (а не принципно като при анализа) описание на класовете
- CRC картата на всеки клас таблица с описание на името, функционалните задължения (responsibility – заданията които изпълнява + контекста им) и списък клобориращи класове за изпълнение на тези задължения
- пример за OPS от 5.10: CRC-карти на RegistrationPage и RegistrationController – фиг. 5.15

Фиг. 5.15 – CRC-карти на RegistrationPage и RegistrationController

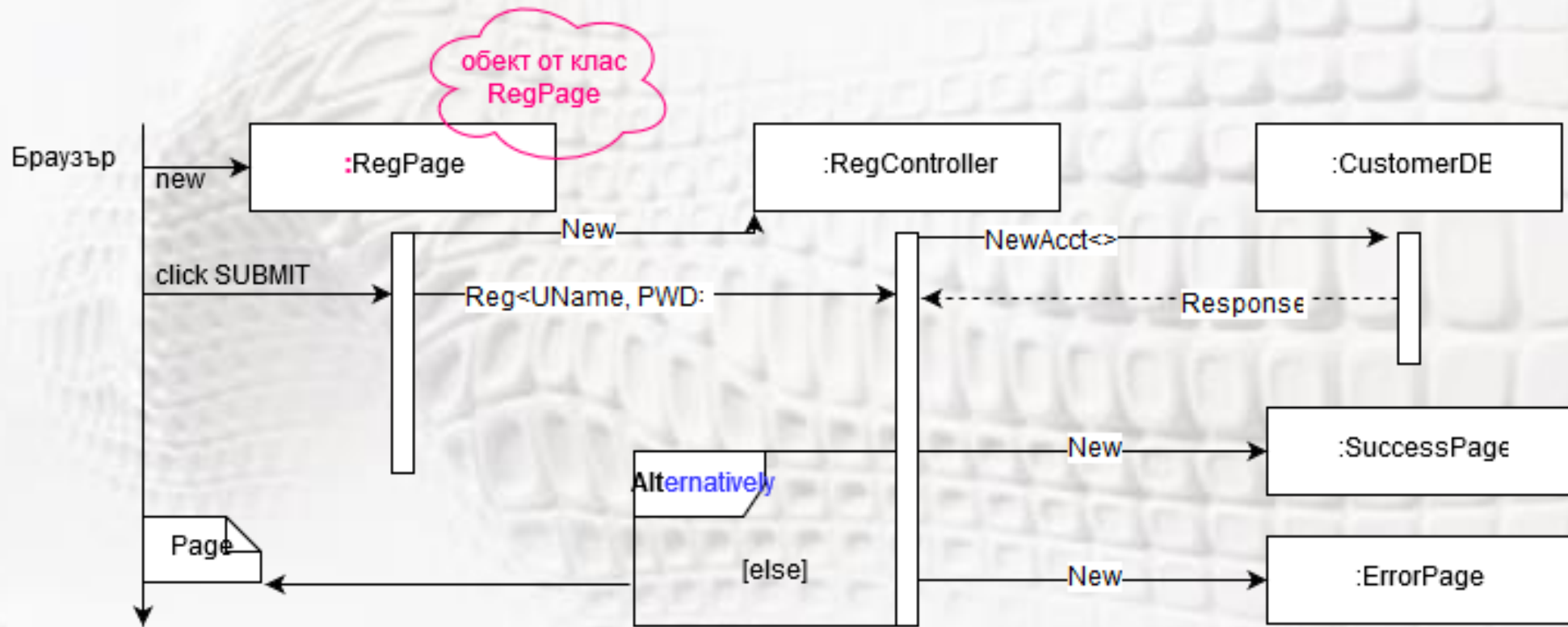
Registration Page	
Responsibility	<u>Collaborators</u>
Знае UserName	
Знае PassWord	
Знае RegistrationControler	
Обработка click-събитие на бутона <SUBMIT>	<u>RegistrationController</u>

Registration Controlle	
Responsibility	<u>Collaborators</u>
Знае RegPage	<u>CustomerDB</u>
Създава Reg SuccessPage	<u>Reg SuccPage</u>
Създава RegErrorPage	<u>RegErrorPage</u>
Обработка регистрацията(т.е. създава запис в DB с Username и Password)	

ОО-проектиране – стъпка 2.

- описва се взаимодействието между обектите от ст. 1.
- прилагат се диаграми на последователността или на комуникациите
- моделът се състои от последователни стъпки, описани чрез обмен на съобщения
- пример – диаграма на последователността за случая Registration – описва обмена между класовете RegistrationPage и RegistrationController – фиг. 5.17:
 - в горната част на диаграмата са взаимодействащите обекти – с означения <object_name>:<class_name> (името на обект може да отсъства)
 - връзките отразяват дейностите на съответните обекти и носят съответните етикети – включително new за създаване на обект от клас-колаборатор
 - в примера само обектите successPage и failurePage са именувани – за разлика от останалите класове – тъй като се предават алтернативно от RegistrationController към RegistrationPage

Фиг. 5.17. – диаграма на последователността за случая Registration (описва обмена между класовете RegistrationPage и RegistrationController)



ОО-проектиране – стъпка 3.

- ст. 3. описва динамичното поведение на по-сложните класове за целия им цикъл на живот – напр. контролните класове – с диаграми на машината на състоянието
- ДМС се извлича от диаграмите на случаите, в които участва дадения клас
- в ДМС отделните състояния означават стабилност на колекцията от променливи на средата и от вътрешни променливи на класа
- вътрешните променливи на класа обикновено задават граничните стойности, с които се сравняват променливи на средата (условие за преход между състоянията на класа) и евентуално се изпълнява преход в друго състояние
- за по сложните класове ДМС е съставна – включва и sub-state диаграми, но:
 - [правило] сложният клас е желателно да се представи от няколко класа ако логическата му функционалност не се описва от едно просто изречение; това се отразява обратно и в CRC-модела

ОО-проектиране – стъпка 4.

- ст. 4. е подробното описание на интерфейсите на всеки клас – изброяват се атрибутите и операциите и тяхната публичност (с + и - в UML)
- публичната част от интерфейса е фиксирана и не трябва да се променя в следващата след проектирането фаза – имплементацията
- публичният интерфейс се състои главно от дефинирани константи и операции:
 - операциите в публичния интерфейс са 4 категории
 - конструктор
 - деструктор
 - аксесор
 - мутатор
 - определянето на публичните атрибути (константи) се базира на следните фактори
 - какви са външните стойности, които класът използва в своите операции – от CRC-диаграмата – напр. класът RegistrationPage използва Име и Парола (5.17)
 - какви са възможните състояния на класа от ДМС – те се включват като атрибути (но обикновено частни)
 - от мощността на асоциациите: 1..1 асоциация изисква скаларен атрибут-указател към асоциирания клас, а 1..* асоциация – атрибут-колекция (вектор)
 - други атрибути, необходими за изпълнение на операциите – обикновено са локални

Обхват на OO-архитектурите

➤ предимства:

- непосредствена връзка с потребителските сценарии и проблемната област
- взаимстване (reuse) и капсулиране на имплементацията
- лесно допълване чрез полиморфизма и класовете-деривати
- устойчивост на системата поради защитеност на локалните атрибути
- удобен преход към други модели и най-вече към компонентна архитектура

➤ възможни проблеми:

- непредвидени странични ефекти при взаимодействието на много обекти, включително при асоциации 1..*
- интерфейсите и вътрешната имплементация на класовете – макар и продукт на отделни фази – не са толкова разграничени, колкото при компонентните архитектури; обикновено се разработват съвместно, което снижава нивото на абстракция (и сложност) на цялата архитектура, а също обичайно води до по-фина грануларност в сравнение с компонентните архитектури
- наследствеността между класовете често води до грешки в спецификацията и следва да се прилага мн. внимателно

Потокови (Data Flow) архитектури

- представят обработката като последователност от трансформации (т.е. групи операции) върху последователност от набори структурирани еднотипни данни
- системата се декомпозира на функционални модули или подсистеми – паралелизъм по управление – аналогия с [нелинейните] конвейри
- интерфейсът между модулите може да е във формата на потоци (streams), файлове, канали (pipes, асинхронни потоци) и др.
- основният паралелизъм е **по данни**, тъй като ритъмът на обработка се задава от наличието на данни за обработка
- по тази причина – отсъствието или минимизирането и импликацията на контролния поток – ПА са подход и стил, приложим предимно при автоматизирани процеси на обслужване – напр. езикови компилатори, автоматизирани системи с пакетно обслужване като разпределените транзактивни системи, вградените системи

Категории потокови архитектури

- топологията на пренос на данните между модулите се задава експлицитно с блок-диаграми (5.23)
- обработката е асинхронна
- модулите поддържат само интерфейс по данни, не и контролен интерфейс и не се адресират взаимно – адресацията е само чрез предаваните данни
- по механизма на свързване между модулите (т.е. на обмен) се разграничават
 - пакетна обработка (Batch Sequential)
 - филтрирани канали (Pipe & Filter)
 - контролни процеси (Process Control)



Пакетна обработка (Batch Sequential)

- най-старият модел на СА за обслужване в транзактивни системи и класическите ОС със стандартен файлов IO и редиректори
- приложението е скрипт с команди за изпълнение на съответните модули в UNIX, DOS, Tcl/Tk – напр. :

```
print myShell.sh
  exec  searching  kwd  <inputFile  >matchedFile          //①
  exec  counting  <matchedFile  >countedFile             //②
  exec  sorting  <countedFile  >reportFile                //③
```

- ① от `inputFile` записва в `matchedFile` само редовете, съдържащи ключовата дума `kwd`
 - ② брой `kwd` във всеки ред на `matchedFile` и записва бройката в началото на всеки ред на `countedFile`
 - ③ сортира редовете на `countedFile`
- този стил е приложим и в съвременните OO езици, където отделните обработващи модули, входът и изходът се представят като методи и атрибути на класа

Приложимост на пакетната обработка

- данните (включително междинните резултати!) са оформени в пакети – файлове, т.е. с **последователен** достъп
- модулите се представят като програми, които се активират със скрипт или като резидентни модули, които сканират входните си файлове
- **неприложима** СА за **интераткивен** интерфейс
- широко приложение за асинхронни паралелни процеси – данните се декомпозират като множество входни файлове, а обработващите модули се репликират в множество възли – принцип на обслужване в пакетната фонова обработка – **Condor, Boinc, SETI@home**

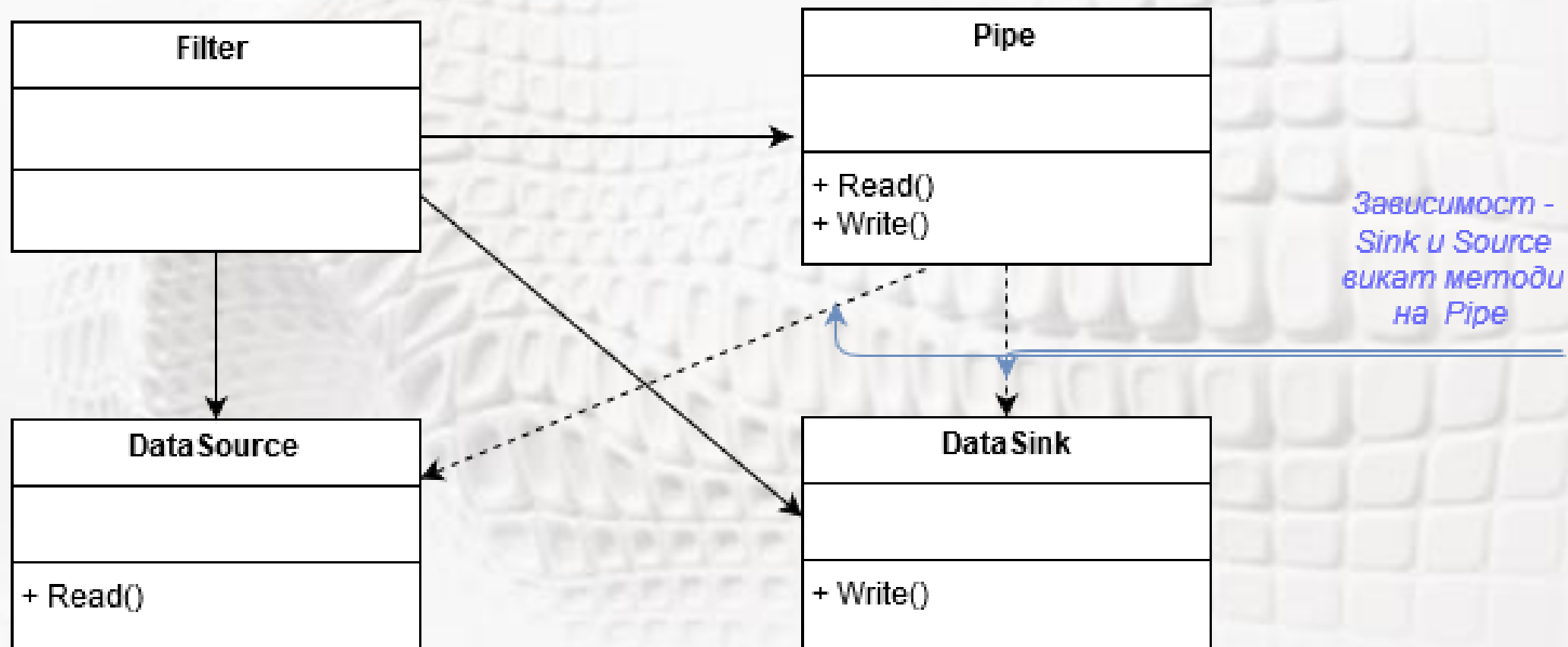
Филтрирани канали (Pipe & Filter SA)

- приложението се декомпозира на източник на данните, филтри, канали (pipes) и консуматор на данните (sink)
- данните са последователни FIFO потоци (буфери, опашки) от байтове, символи или записи, които представят в последователен вид всички структури – вкл. и по-сложни, които се сериализират – в ОС marshalling/unmarshalling
- филтрите
 - трансформират потока данни – без необходимост да изчакват готовност на целия пакет за разлика от пакетната обработка!
 - записват изходните данни в канал, който ги предава на друг асинхронно работещ филтър
 - 2 типа филтри:
 - активен филтър – изпълнява операциите pull/push върху пасивни канали – каналите осигуряват съответните операции, а инициативата е на филтъра. В Java PipedWriter и PipedReader класовете предоставят този интерфейс за канали
 - пасивен филтър – предоставя push/pull интерфейси на каналите
- каналите преместват – а по същество съхраняват – потока данни, които се обменят между два филтъра

Свързаност на филтрираните канали

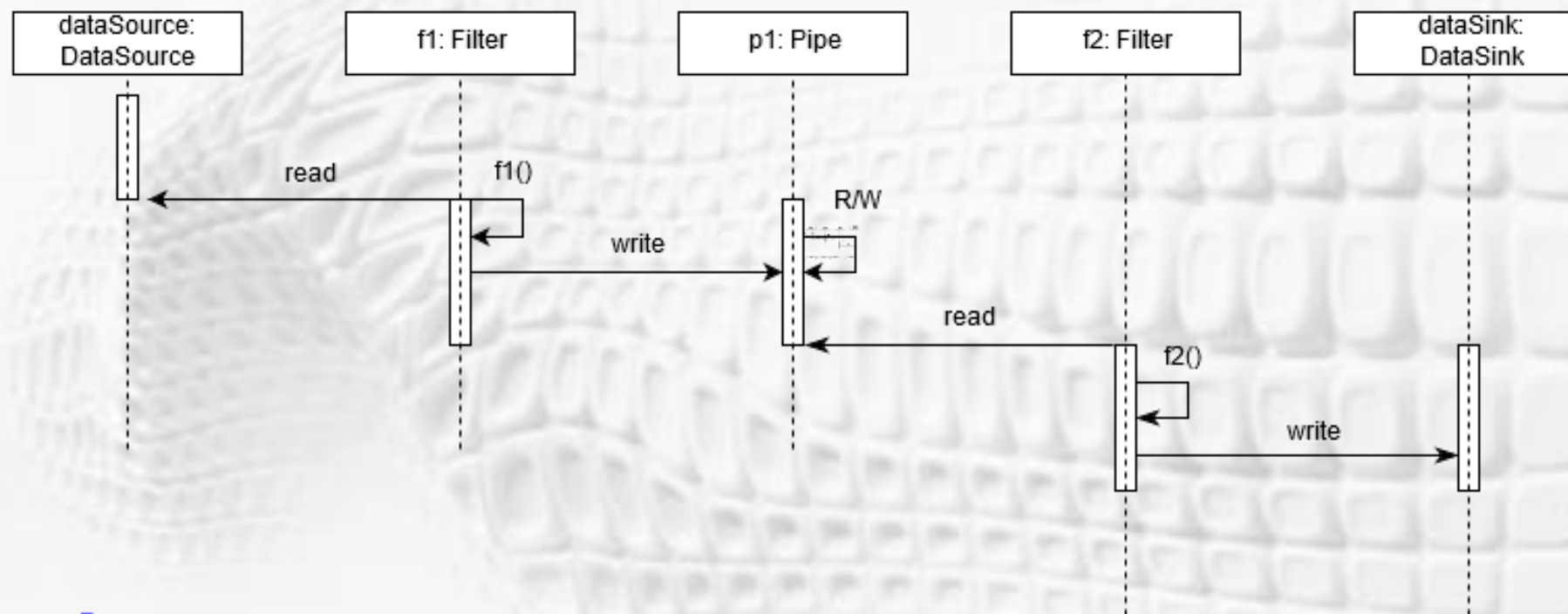
- клас-диаграма на СА с филтри и канали 5.27.1 – активният модул е с плътни интерфейсни линни
- филтърът е свързан с до 3 класа – източник на данните, консуматор и канал
- блокова и последователностна диаграма на ФКСА – 5.27.2
- ФКСА се организира лесно в пакетните ОС –
 - напр. в Unix `who | wc -l` означава пасивен канал между две операции – в случая `who` генерира списък от потребителите, `wc` брои думите в списъка (спрямо стандартни разделители); поддържат се канали с имена, а филтри могат да са произволни процеси в основен и фонов режим (`fore-` и `background`)
- макар, че управлението на обработката е по данни, паралелизмът е по управление (MPSD) и архитектурата е приложима когато обработката може да се раздели на асинхронни модули
 - това означава, че се проектират няколко [балансиращи] програми, а данните не се декомпонират
- реализира модела производител/консуматор
- не се поддържа динамичен и интерактивен интерфейс – ограничение, което е предимство при дадени приложения
- приложението се ограничава от формата на данните в каналите – обикновено се използва ASCII код

Фиг. 5.27.1.

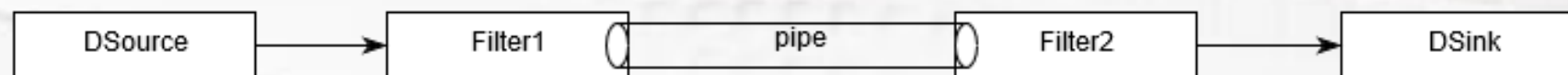


Фиг. 5.27.2.

Последователна
диаграма



Блокова

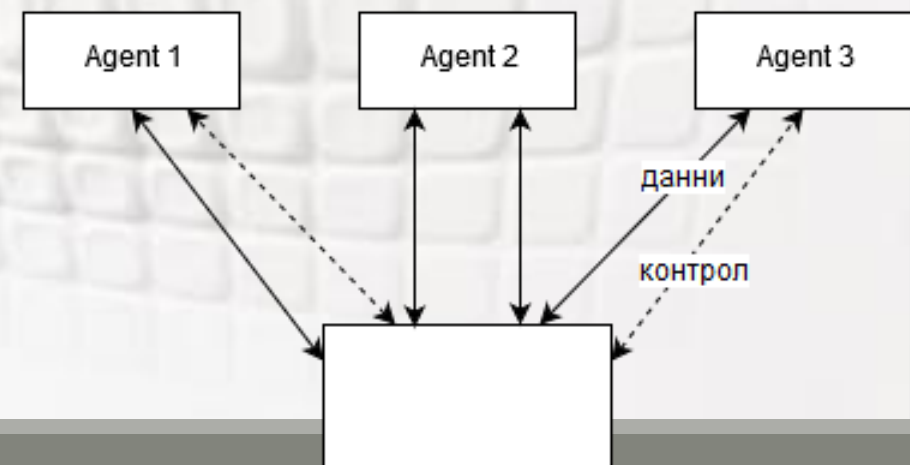


Контролни СА

- прилагат се при вградените системи (ВАС) – компютърно контролиране на процеси в реално време с или без човеко-машинен интерфейс
- при вградените системи управлението е на база на сканиране на променливи на средата, извлечени като поток данни от сензори и управляващо въздействие чрез компютърно контролирани актуатори – напр. автомобилен ABS – 5.30
- и при КСА процесът се разделя на няколко модула, но те са от 2 типа
 - контролни модули – за следене и манипулиране на променливите на средата и състоянието
 - изпълнителски модули – за управление на актуаторите
 - връзките между модулите са чрез поточни данни
- типове контролни потоци при КСА
 - контролирани променливи – характеристики на ВАС (сила на ток, налягане и др. физически контроли на изпълнителните актуатори) – те се измерват текущо от сензорите и се съпоставят с контролните константи т.е. целевите стойности
 - входни променливи според проблемната област (скорост, налягане, температура, влажност, GPS координати)

Контекстни архитектури (Data Centric)

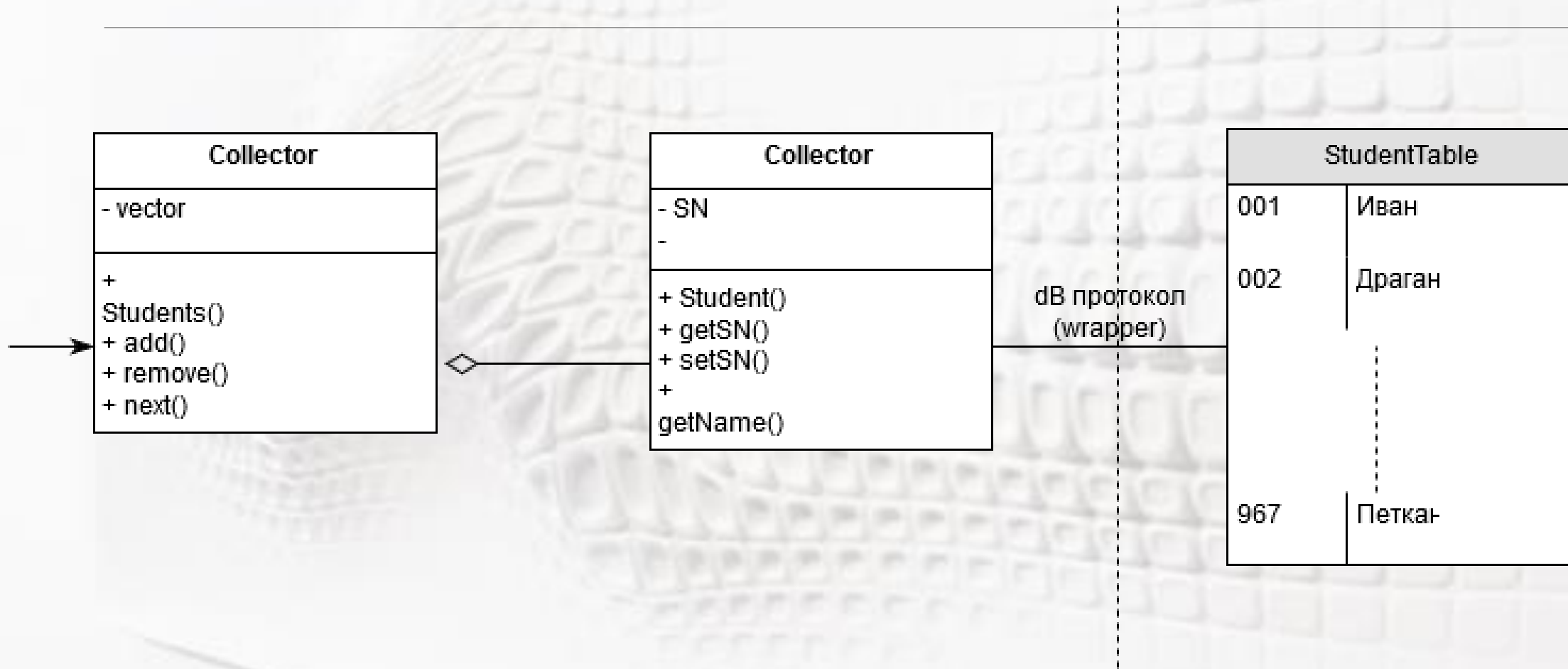
- ✦ характеризират се с централизирано хранилище на данните, които са достъпни за всички компоненти на системата, така че декомпозицията е на модул за управление на достъпа до данните и агенти, които извършват операции върху тях
- ✦ интерфейсът между агентите и данните може да е явен – напр. RMI или RPC – или имплицитен – напр. транзактивен
- ✦ в чист вид KArch не предвиждат преки комуникации между информационните агенти – [фиг. 5.31](#).
- ✦ модулът данни изпълнява операции по извличане или регистриране и промяна на записи – по 2 възможни модела:
 - ✦ **хранилище** (repository) – с активни (инициативни) агенти – хранилището е обикн. е организирано като СУБД, CORBA, UDDI или Web-услуги
 - ✦ **черна дъска** – с инициатива на модула данни – агентите са абонати за събития (event listeners), които настъпват при промяна в данните и на които абонатите отговарат реактивно – често при AI-разпределени приложения, охрантелни системи за разпознаване на звук и образ, системи за управление на бизнес ресурси – складове, транспорт



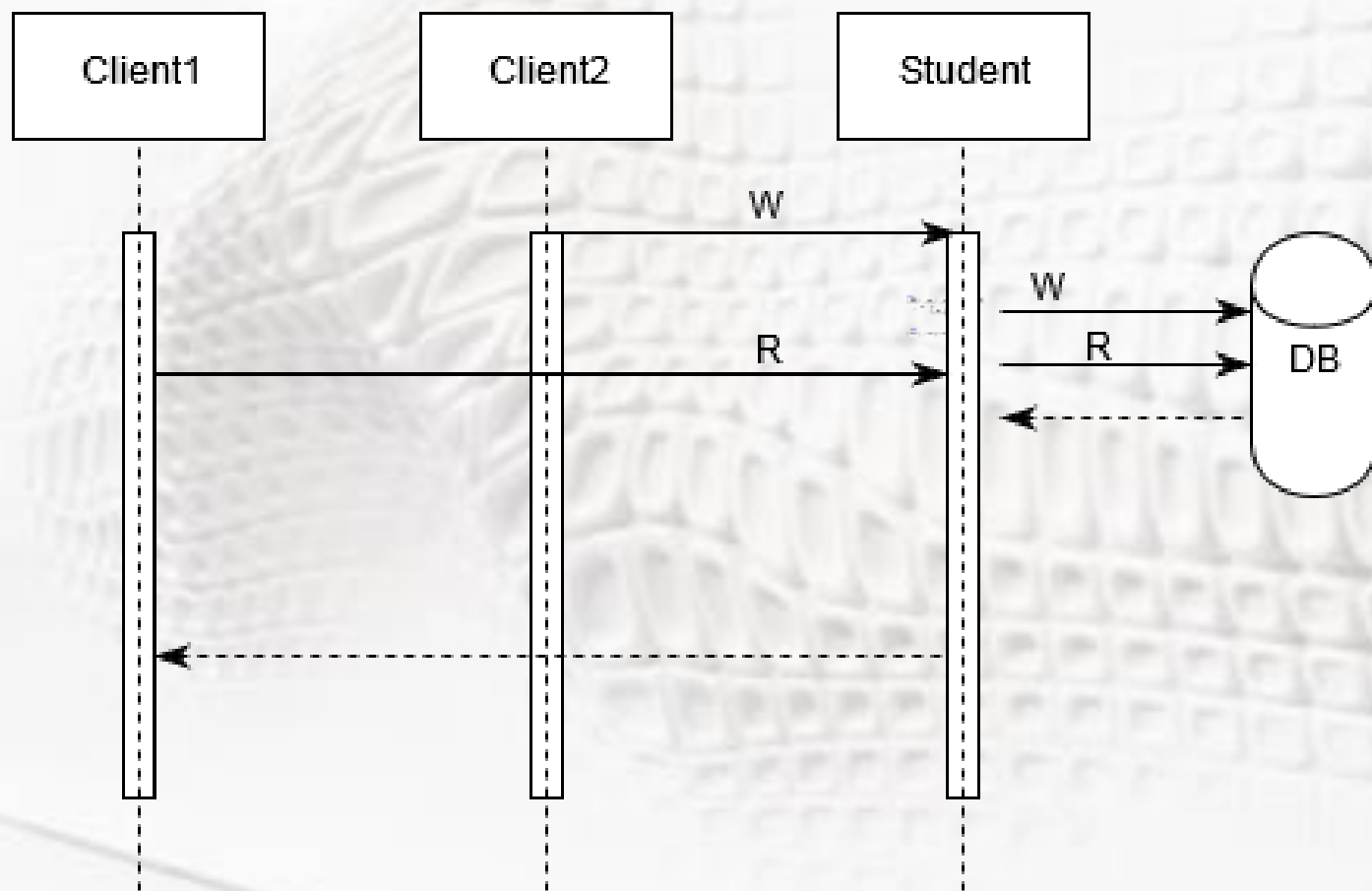
Контекстни архитектури с хранилище

- Макар и с управление по данни – за разлика от потоките архитектури за пакетно обслужване на транзакции – тези архитектури поддържат интерактивните [U]I
 - пример: клас-диаграма на университетска информациялна система – 5.32.1
 - класът Collector поддържа вектор на колекция от студентски записи и затова агрегира клас Student, като поддържа UI за извличане, добавяне и промяна на записите за студентите
 - класът Student е интерфейс към таблицата на студентите, чиито инстанции представят по един запис (т.е. ред) в нея
 - диаграмата на последователността 5.32.2 представя споделянето на данните чрез класа Student между няколко клиенти
- релационните СУБД са обичайната платформа за имплементация на тези архитектури, тъй като поддържат свързаност (консистентност) на разпределения достъп до данните, както и множество системни средства за операции, базирани на метаданни
- за по-висока отказоустойчивост и защита на данните се прилагат разпределени хранилища
- основен недостатък е статичната структура на данните – еволюция в структурата на релационните таблици се прилага трудно, струва скъпо и надеждността ѝ се проверява трудно

Фиг. 5.32.1.



Фиг. 5.32.2.



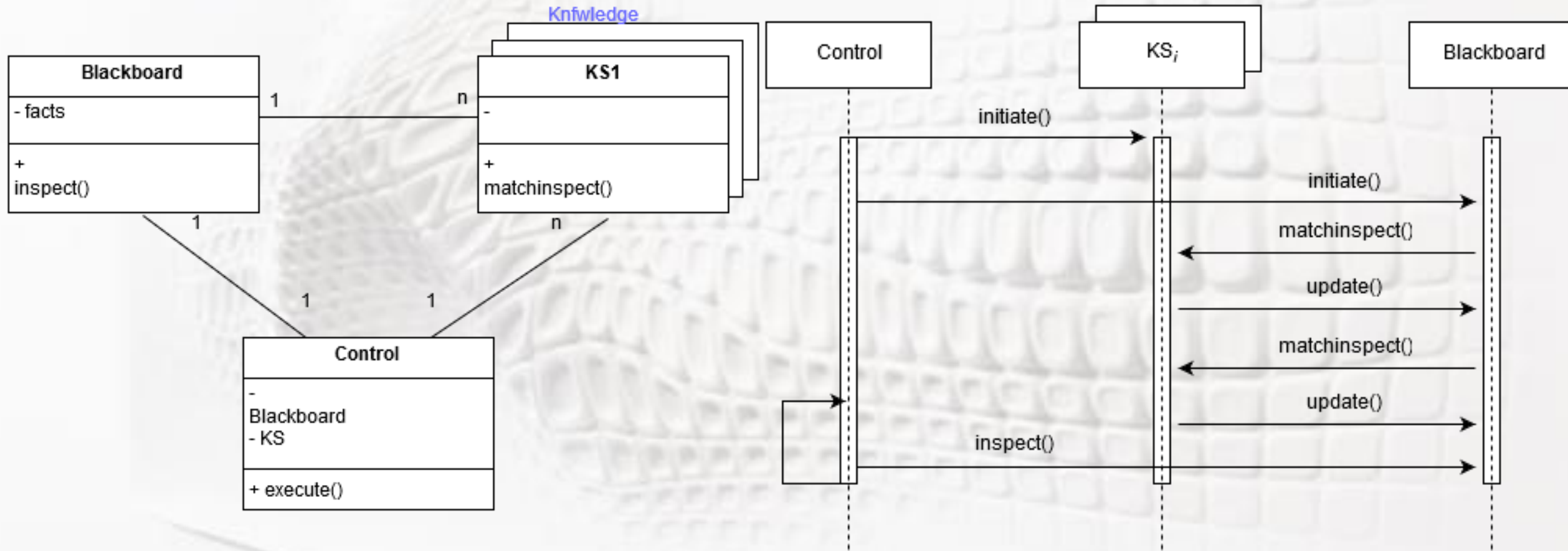
Контекстни архитектури с черна дъска

- ориентирани са главно към проблеми, решими с методите на AI – най-вече разпознаване на шаблони в различни области (първите приложения от края на 1970те са експертни системи в метеорология, изображения, звук, молекулярна химия)
- декомпозират решаването на проблеми също на два[+] дяла
 - **черна дъска**, съхраняваща данни – факти и хипотези т.е. еволюционни модели над фактите
 - **източници на знания** – паралелно работещи агенти, които съхраняват различни страни (данни, организирани като знания) от проблемната област – всеки ИЗ капсулира специфичен аспект от проблема и е отговорен за частни хипотези и решения като част от общото решение
 - [**контролер** – система за начално зареждане и управление на разпределеното приложение]
- запазва се блок-д-та от 5.31., но контролният поток е само от ЧД към ИЗ:
 - неявни (имплицитни) обръщания към регистрираните в ЧД агенти-източници
 - обръщанията възникват при промени в данните и се предават към абонираните за тези промени ИЗ, които изпълняват реактивно заложените в тях логически правила за извод
 - този асиметричен механизъм на обмен е известен като модел publish/subscribe (pub/sub) в общите комуникации
 - класифицират се като слабо-свързана (loosely coupled) РС поради асинхронния комуникационен модел с обмен на публикувани съобщения към абонатите (за разлика от силно свързаните (tightly coupled) системи с хранилища, където транзактивното обслужване е свързано със заключване на данните за конкурентен достъп

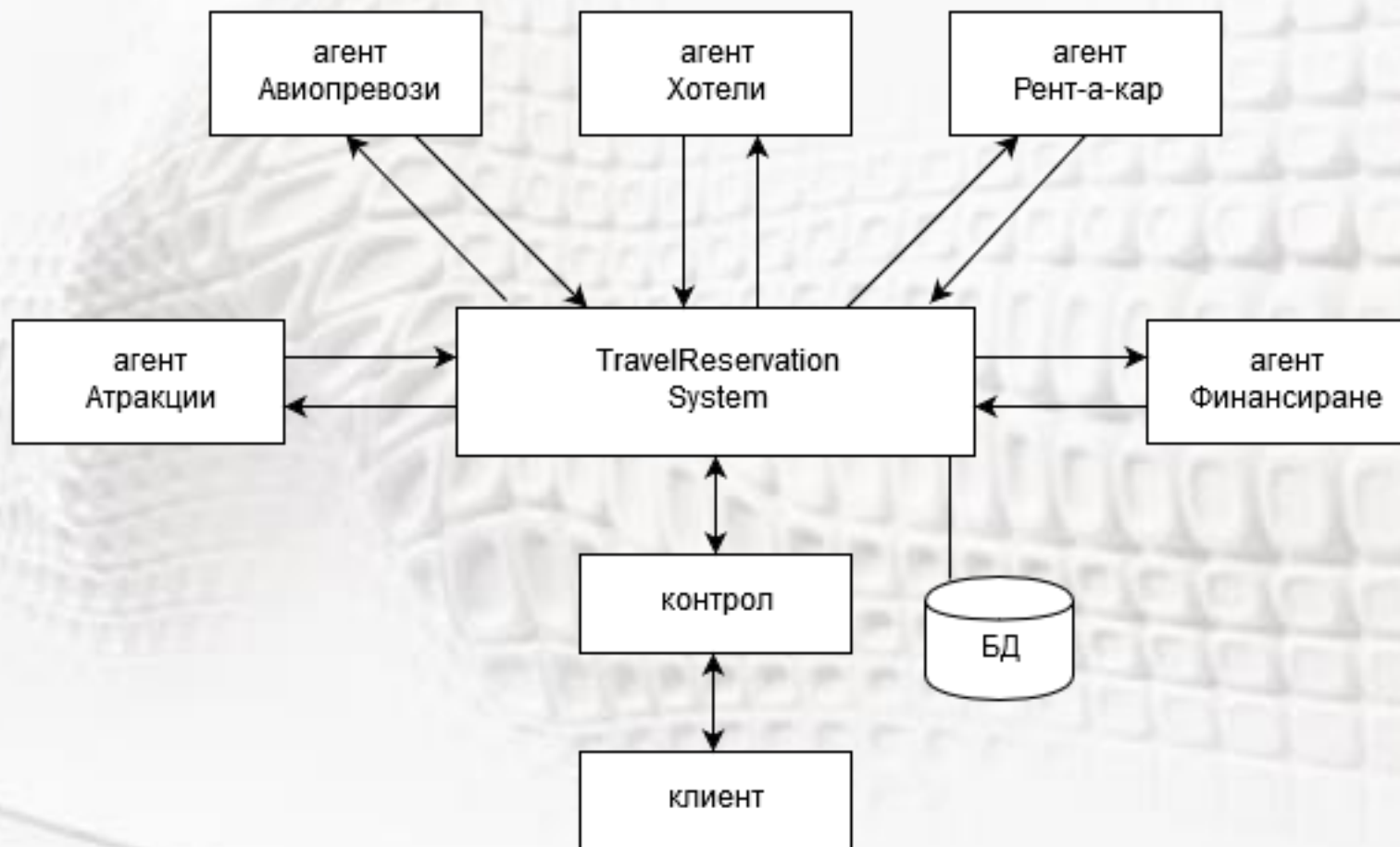
Диаграми на КАрх с черна дъска

- ✦ клас-диаграма на такава архитектура – [5.36.1](#)
 - ✦ класовете-източници KnowledgeSource съхраняват специфичните правила за логически изводи, регистрират се в съответната ЧД, абонира се за оповестяване на промени в данните на ЧД и евентуално генерират реакции с изменения в локалния си или общ (ЧД) контекст; форматът на знанията и правилата за всеки ИЗ може да е специфичен
 - ✦ ЧД управлява общия контекст, регистрира промените в него, оповестява абонатите и регистрира евентуалните реакции, както и съхранява крайното решение
 - ✦ контролерът инициира ЧД, множеството на ИЗ, инспектира състоянието им и публикува крайното решение
- ✦ последователностна диаграма на архитектурата – [5.36.2](#)
- ✦ блок-диаграма на КАЧД на система за туристически консултации – [5.36.3](#)
 - ✦ обединява множество резервационни агенции – пътни, хотелски, за атракции, за коли под наем, кредитни и т.н.
 - ✦ клиентските заявки се публикуват на ЧД и се оповестяват съответните агенти, чрез реакциите на които се изготвят един или повече планове за туристическо пътуване и съответното финансиране
 - ✦ всички операции се инициират по данни, а се поддържа и UI: типично за КАЧД клиентският интерфейс през контролера е минимален – примерно еднократен, но интерфейсьт за управление на агентите може да е итеративен

Диаграми на КАрх с черна дъска – фиг. 5.36.1. и 5.36.2.



Диаграми на КАрх с черна дъска – фиг. 5.36.3.



Обхват на КАрх с черна дъска

- подходяща архитектура за комплексни неизследвани и особено мултидисциплинарни проблеми които са
 - без детерминистично решение и с представяне на контекста във форматите на AI
 - неподходящи за търсене на решение с пълно обхождане на проблемния домен поради изчислителната сложност или непълнота/неточности в данните
- може да се генерират оптимално или няколко субоптимални решения или решения на частни подпроблеми
- за разпределена обработка с умерена скалируемост поради централизирания контекст
- проблем е еволюцията в структурата на контекста поради обвързаност с агентите на знания
- отсъствието на междуагентни комуникации води до необходимост от централизираната им синхронизация (например приоритетна) на достъпа до общия контекст
- трудно се формулира условие за край на обработката поради недетерминистичния характер на проблемите