РАЗПРЕДЕЛЕНА БИЗНЕС ЛОГИКА В ИНФОРМАЦИОННИ СИСТЕМИ. ИЗГРАЖДАНЕ НА КОНТЕЙНЕР ЗА АНАЛИТИЧНА БИЗНЕС ЛОГИКА

Александрина Мурджева*

1. ВЪВЕДЕНИЕ

Съвременната трислойна архитектура на приложенията позволи да бъдат използвани разнообразни технологии във всеки един от нейните слоеве и породи много и различни спорове за "местата на нещата". Всеки от слоевете беше натоварен с определена задача, но все по-често може да се видят в практиката интересни спекулации и употреби.

Един от споровете, който се води, е "къде е мястото на бизнес логиката (БЛ) в едно приложение?". Причината за този спор е в това, че и средният слой (middleware), и често сървърът (базата данни) в многослойната архитектурата са осъществени с технологии, еднакво добри кандидати да поемат реализацията на бизнес логиката.

Съвременните среди и езици за реализация (.NET, J2EE, PHP) са достатъчно мощни средства, но и съвременните бази от данни (MS SQLServer, ORACLE) с процедурните си възможности също предоставят на разработчиците много добри инструменти.

Къде се поражда проблемът и защо възниква изкушението за бизнес логиката да се търси алтернативна реализация?

Целта на настоящата студия е да се направи анализ на причините, довели до тази дилема в областта на изграждане на информационни системи, и да се представи предложение за конкретно решение под формата на система за класификация и на концепция за архитектурен елемент. Двете нови предложения, които се правят в настоящата студия, са резултат от обобщение на резултати при прилагането на различни технологии и наблюдението на приложения, разработени с различни интерпретации на технологични идеи.

Системата за класификация е насочена към предлагане на набор от критерии, чрез които да се оцени и вземе решение за това къде е най-подходящото място за реализацията на бизнес логиката на приложението. Предлаганият нов архитектурен елемент, наречен от автора ***контейнер за аналитична бизнес логика***, има за цел да се предложи концепция и насоки за реализация на нов елемент в многослойната архитектура, с помощта на който да се абстрахира максимално определена част от бизнес логиката с цел тя да може да бъде използвана прозрачно и многократно.

Поставените цели имат както теоретичен, така и практически характер. В теоре-

* Александрина Мурджева е доктор по икономика, главен асистент в катедра "Информационни технологии и комуникации" в УНСС; тел: 088 852 28 80, e-mail: amurdjeva@abv.bg.

тичен план студията ще направи обобщение на текущия опит при изграждане на многослойни приложения, като акцентира върху актуалните проблеми, свързани с поддръжката на приложения и тяхното развитие. А от гледна точка на практиката, за дефинираните проблеми ще бъдат направени конкретни предложения за преодоляването им.

2. СЪЩНОСТ, СТРУКТУРА И РЕАЛИЗАЦИЯ НА БИЗНЕС ЛОГИКАТА

2.1. Реализация на бизнес логика

Понятието бизнес логика (БЛ) все още не е получило своята точна дефиниция, която да определи същността ѝ и да бъде изходна позиция за разбирането и реализацията ѝ. Едно от съществуващите определения на БЛ е, *че това е не техническо понятие, описващо функционални алгоритми за размяна на информация между база данни и потребителския интерфейс* [11].

Това определение е недостатъчно и не разкрива пълната същност на БЛ. Необходимо е да се даде определение, което да покаже както формалната, така и логическата страна на понятието, за да може да бъде направен анализ на структурата и семантиката на БЛ.

Формалната страна на БЛ е свързана с нейното представяне като алгоритми, а логическата е свързана с това, което тези алгоритми трябва да изразят.

2.1.1. Определение за бизнес логика

Един опит за даване на ново определение е:

Бизнес логиката може да бъде определена като набор от бизнес процеси, които едно приложение трябва да формализира и автоматизира. Формализацията на тези бизнес процеси е набор и последователност от алгоритми, представени на избран език за програмиране. Тези алгоритми управляват (извършват обработки върху) съответен набор от данни, представен със средствата на база от данни.

По отношение на разбирането за същността на БЛ не съществуват много мнения и горното определение синтезира класическото виждане за нея.

В съвременната многослойна архитектура на приложенията класическото схващане за БЛ има и класическа реализация: *представянето на данните и тяхното съхранение е задача на базата данни (сървъра в архитектурата), а поддържането на данните и БЛ е в средния слой на архитектурата.*

Когато става дума за приложения, поддържащи данни, казано най-общо, а това е значителната част от съвременните бизнес приложения, възниква въпросът “на кого е присъща задачата да реализира бизнес логиката?”.

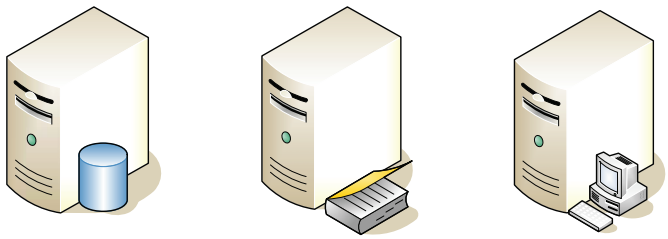
Както става ясно от даденото определение, бизнес логиката обхваща алгоритми и данни в най-общ вид. Реализацията на двете също има свои класически форми като езици за програмиране, алгоритми и място в многослойната архитектура.

В практиката има два крайни варианта за реализация, всеки от които има своите

предимства и недостатъци. Те ще бъдат обобщени по-долу. Но по-интересно е търсенето на баланс между тези две крайности, което е един от изходните пунктове за анализ в настоящата студия.

2.1.2. Бизнес логика в средния слой (middleware)

Реализацията на БЛ в средния слой на приложението има своите предимства и недостатъци. При тази реализация почти цялата бизнес логика се съсредоточава в средния слой, като слоят на базата данни се ангажира единствено със съхранението и с минимален контрол върху данните (Фиг. 1).



Фиг. 1. Класическа реализация на бизнес логика

Предимствата са безспорни и масовият избор на този подход при изграждане на приложения ги доказва.

Като такива предимства могат да бъдат посочени [7][13]:

- Пълното освобождаване на потребителския интерфейс от обработката на данните, което дава големи възможности за избор на алтернативи за реализация му.
- Пълно освобождаване на бизнес логиката от презентационни задачи, което дава възможности за избор на алтернативи за реализация на тази част от приложението.
- Възможност за разпределяне на програмния код и използване на различни технологии за реализация на всяка една част.
- Цялата бизнес логика е реализирана на едно място и лесно може да бъде проверявана и коригирана.
- При реализацията се използва мощен език за програмиране.
- Базата данни приема ролята само на хранилище за данни и акцентът при нея пада върху ефективното съхранение и извличане на данни, без реализация на сериозни бизнес правила за проверка на данните или натоварването й с визуалното им представяне.
- Бързо и потенциално евтино изграждане на приложения чрез използване на съществуващи, тествани компоненти от бизнес логика и за достъп до данните.
- Областта на промените е потенциално ограничена, т.е. промените в един от слоевете не засягат останалите слоеве.

Като две страни не една монета недостатъците на този подход съществуват и именно те са причина в практиката да се срещат различни форми на прилагане на

многослойните архитектури чрез комбиниране на технологии и прилагането им в различни слоеве на приложението.

В три посоки обаче е интересен анализът на недостатъците на този подход, а именно: **поддръжката, настройването и използването** на реализираната БЛ. Следващият анализ акцентира върху информационни системи, свързани с обработка на данни, което е значителната част от информационните системи днес.

2.1.2.1. Поддържане на бизнес логика

Поддържането на програмния код в приложението е свързан с прекомпилиране и създаване на нови версии на програмите. Оперативната поддръжка на работещи приложения е трудна, тъй като тя се извършва непрозрачно за потребителите и със сериозната намеса на разработчиците. Поддържането на програмния код е част от жизнения цикъл на една система и появата на необходимост от това не е недостатък на дадена технология.

По-интересен е анализът *коя част от програмния код и кога поддръжката му* водят до затруднения и нарушаване на нормалния режим на работа на системата. В приложения, обработващи данни, програмният код или бизнес логиката може най-общо да се раздели на такава, свързана с манипулиране на данните, и свързана с алгоритмизацията. Първият тип програмен код най-често се реализира с естествения език за обработка на данни SQL, а вторият тип – с помощта на избран език за програмиране.

Поддръжката на програмен код на компилируем език за програмиране е неизбежна. Но как стои въпроса с поддържането на програмния код за поддържане на данни, който е интегриран в алгоритмите? Поддържането на SQL код в базата данни е много по-естествена задача и е свързана с много по-голяма степен на прозрачност за потребителите. В противовес на това интегрирането на SQL код в алгоритмите увеличава в значителна степен сложността при тяхната поддържа.

2.1.2.2. Настройване на приложения

Настройването на приложенията е задача, която е пряко свързана с поддържането на приложенията. По отношение на настройването на бизнес логиката на приложението има два основни аспекта – настройване на самата логиката и подобряване на производителността. Подобряването на производителността на приложения днес има особено важно място и се превръща в проблем с много висок приоритет при разработката на информационни системи.

Други две много важни характеристики на приложенията поставят още по-силно проблема с възможностите за настройване на приложенията. От една страна, **оперативността на приложенията** във все повече предметни области става критична, т.е. времето е скъп и недостатъчен ресурс, и заедно с това обемите на данните, върху които трябва да се осигури тази оперативност, нарастват значително. Подобряването на производителността в случаите, когато БЛ е изцяло в приложенията, е трудно, защото коригирането на интегрираната SQL логика в кода на приложението винаги е свързано с действия, непрозрачни за потребителя и със задължителното участие на разработчиците.

От друга страна, в практиката се наложи модел на създаване и поддържане на приложения, при който разработката на приложения и **администрирането** на средата, в която те работят, се обособиха като отделни отговорности. Все по-често потребители, които са заинтересовани информационната система да работи оперативно, поемат задачата с администрирането и настройването на средата за работа на приложението. В този смисъл и дейностите, свързани с подобряването на производителността, са в техни ръце. Но при добре "скрития" и/или интегриран в недиректен вид SQL код (напр. както това става при използване на Hibernate [14]) изпълняването на тази задача е невъзможно.

2.1.2.3. Използване на реализирана бизнес логика

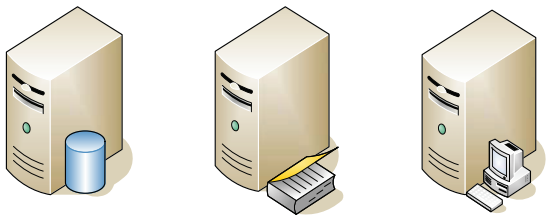
Не е нова тенденцията при разработването на приложения за бизнеса или за други области да се използват съществуващи програмни фрагменти като изолирани части от целия програмен код или като завършени цялостно модули. До скоро това се прилагаше най-често за елементи от приложенията, достатъчно абстрактни и необвързани с конкретна предметна област.

Днес новият аспект на тази тенденция е именно в това натоварени с бизнес смисъл компоненти да могат да бъдат използвани многократно ([17]). Постигането на това е сериозно предизвикателство и е допълнителна задача при разработка на приложения и по-специално при реализацията на бизнес логиката на приложението, изискващо да се отдели специално внимание на потенциалните елементи от програмата, които са добри кандидати за многократно използване. Това е нова и отделна задача за разработчика.

По-интересно е, да се разгледа същият този проблем в рамките на самото приложение. Как съществуващите алгоритми, реализирани чрез SQL код, да могат да бъдат използвани многократно в рамките на същия бизнес проблем, дори и извън приложението? Конвенционалното интегриране на SQL код в приложенията не позволява той да може да бъде използван извън него, респ. този код е тясно обвързан с конкретното приложение и на практика е неизползваем многократно.

2.1.3. Бизнес логика в базата данни (server)

Алтернативата за реализация на БЛ е използването на силните процедурни възможности на съвременните бази данни (например като съхранени процедури и тригери). В този случай базата данни се натоварва не само с поддържането на данните и формалния контрол върху тях, но и със специфичната логика по отношение на автоматизирания бизнес процес (Фиг. 2).



Фиг. 2. Бизнес логика в база данни

Аналогични разсъждения могат да бъдат направени и тук по отношение на трите важни аспекта – поддържане, настройване и използване.

Както приложните програми са слабо място за реализация и поддържане на програмен код за манипулиране на данни (SQL код), така базите от данни не могат да предоставят сериозни алтернативи на програмните среди по отношение на другите аспекти на програмния код като реализация на алгоритми, валидация на стойности, комуникация с клиентската част на приложението и т.н.

Предимствата при използване на процедурните инструменти в БД, споменавани от много автори [18][16][8][15][6] и доказани в практиката, могат да бъдат формулирани така:

- Съхранените процедури са модулни и скалируеми. Промяната в тях, както и надграждането им, комбинирането им в други програмни елементи е в голяма степен по-лесно, ако същата обработка трябва да се случи в приложението.
- Съхранените процедури могат да се настройват и оптимизират по отношение на особеностите на БД (tunable).
- Съхранените процедури имат по-добро бързодействие от други форми за изпълнение на SQL заявки.
- Съхранените процедури абстрахират функционалност и структури. Използването им позволява приложението да се проектира и реализира като "общ алгоритъм" на бизнес процеса.
- Лесна поддръжка на кода на приложението. Премахва се SQL код от останалите слоеве на приложението и пренасянето им в БД. По този начин се осигурява подобър контрол по време на изпълнение на приложението върху заявките във връзка с коректността им и оптимизирането им.
- Пълна прозрачност за потребителя. Настройването на приложението, внасянето на корекции в бизнес логиката като интерпретация на данни и поведение на алгоритми, се извършва "незабелязано" за потребителя, без да се нарушава стереотипа му на работа.
- Намаляване на мрежовия трафик. Безспорно предимство, свързано с факта, че запитванията се изпълняват на сървъра и няма изпращане на заявки по мрежата, а само на резултати от тяхното изпълнение.

Направеният анализ показва, че абсолютизирането на задачата на всеки елемент на архитектурата на приложението и екстремизирането при неговото използване, не осигуряват постигане на качествен резултат в трите аспекта – поддържане, настройване и използване.

Изводът, който се налага, е, че крайните подходи при проектирането и реализацията на БЛ са резултат от обобщеното разбиране на бизнес логиката.

От друга страна, все по-осезаем става сблъсъкът на разработчиците при изграждането на приложение с невъзможността да подобряват бързодействието прозрачно за потребителя и без да нарушават оперативната работа на системата, както и с необходимостта от многократно използване на реализиран вече код за манипулиране на данни в други схеми на БД или контексти.

Търсенето на решение или поне задаването на посока, в която то може да се намери, е в предлагането на по-различно или по-детайлно разбиране на бизнес логиката, на основата на което да се търсят нови реализационни аспекти.

В следващото изложение ще бъде предложен именно такъв детайлен поглед върху същността на бизнес логиката.

2.2. Структура на бизнес логиката

Внимателен анализ на бизнес логиката на едно приложение, с опит да се абстрахира нейната структура, би помогнало за по-прецизния избор на инструмент за нейната реализация и реабилитация на съхранените процедури като средство за постигане на по-добра организация на програмния код[1].

2.2.1. Признаци за разделяне на елементите

Бизнес логиката не само представя във формализиран вид определен бизнес процес. *Бизнес логиката има своя вътрешна структура*. Фактът, че тя може да бъде разпределяна между слоевете на приложението го показва. Интересен е отговорът на въпроса дали могат да *бъдат дефинирани общи елементи или тази структура е силно зависима от предметната област и конкретния решаван проблем.*

Откриването на елементите на тази структура се нуждае от критерии, по които те да се разпознават. Идея за подобни критерии може да бъде открита в цялостния замисъл на многослойната архитектура. Предлагаме следния набор от основни критерии: вид алгоритъм и транзакционност на обработката.

Един от признаците, по които могат да се формулират елементи на бизнес логиката, е видът на алгоритъма, който се прилага за реализацията му. Алгоритмите са най-общо "алгоритми по памет" или "алгоритми по данни", т.е. къде е акцентът в действието. Под **алгоритми по данни** ще разбираме най-общо *последователност от действия, свързани с извличане и обработване на извлечените от базата данни стойности. Тези алгоритми реализират чести обръщения към базата данни и за тяхната реализация се изисква използване на език за манипулиране на данните.*

Вторият вид **алгоритми по памет** визират процедурни обработки в приложенията, които имат за цел да формализират някаква последователност от технологични стъпки, които използват стойности (независимо как са получени) и създават нови стойности.

Вторият признак е транзакционността на обработката – т.е. дали тя представя оперативно или аналитично действие. Транзакционността на обработката акцентира върху това до колко критичен е резултатът по отношение на оперативната работа и дали обхваща оперативни или исторически данни.

Базата данни (и по-конкретно нейните инструменти за представяне на процедурна логика) е най-добрият кандидат за реализация на аналитичните обработки (нетранзакционните) и предлага много оптимизирани инструменти за работа с данни.

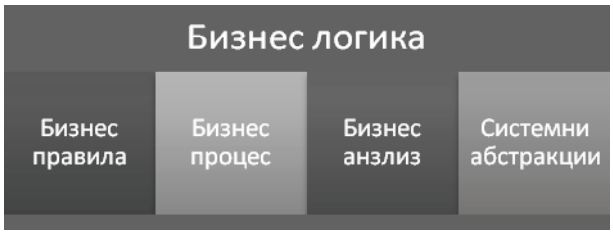
Средите за програмиране, с които са населени другите слоеве на приложението (най-вече средния), са безспорни при реализация на алгоритми по памет и оперативните действия в системата.

	Транзакционна	Аналитична
Алгоритми по данни		База данни
Алгоритми по памет	Среди за програмиране	

Фиг. 3. Избор на средства за реализация на бизнес логика

На основата на горните критерии може да се определи следната структура на бизнес логиката (Фиг. 4):

- **Бизнес правила** – операции по формална и логическа валидация на данните. Най-често са операции, използващи алгоритми по памет и категорично става дума за транзакционни обработки.
- **Бизнес процес** – алгоритъм, използващ валидираните данни за получаване на други оперативни данни. Това най-често са по-сложни алгоритми върху данните, но които са транзакционни.
- **Бизнес анализ** – сложни операции по агрегация на данни и получаване на обобщени аналитични резултати. Използват се алгоритми по данни и това са аналитични обработки (първо и най-ясно тяхно проявление са справките).
- **Системна абстракция** – алгоритми за обезпечаване на сигурност при изпълнение на приложението и среда за изпълнение на горните елементи. Оперативни (транзакционни) алгоритми, по-често по памет.



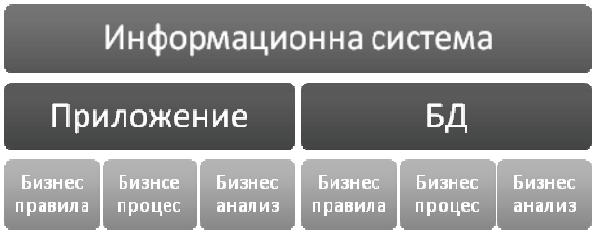
Фиг. 4. Структура на бизнес логика

На Фиг. 5 е представена класификацията на определените по-горе елементи според предложените два основни критерия за идентификация:

	Транзакционна	Аналитична
Алгоритми по данни	Бизнес процес	Бизнес анализ
Алгоритми по памет	Бизнес правила	

Фиг. 5. Класифициране на елементите на бизнес логиката

Всеки от тези елементи на практика е достатъчно обособен и това дава възможност да се търсят подходи за разпределяне на бизнес логиката не като цяло, а за всеки елемент по отделно (Фиг. 6).



Фиг. 6. Разпределение на бизнес логика между слоевете на приложението

2.3. Разделяне на бизнес логиката

2.3.1. Подходи при разделянето на бизнес логиката

Могат да бъдат дефинирани няколко подхода за избора на начин за разпределяне на бизнес логиката. Те се различават, от една страна, по това в каква степен може да се възползва разработчикът от положителни страни, и в каква степен ще се сблъска с отрицателните, а, от друга, по това, кой слой на приложението поема по-голямата тежест при реализацията на бизнес логиката[1].

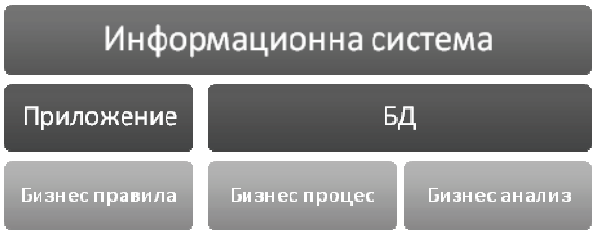
Дефинирането на теза подходи е на основата на обобщение на традиционните подходи в практиката. Именно тук се наблюдават двете крайности в поведението на разработчиците и липсата на добре обмислен и формулиран среден вариант.

В следващото изложение ще се предложат основни формулировки на подходите, започвайки от търсената “златна среда”:

- **Структурен подход.** При него бизнес логиката се разглежда като отделни елементи и за всеки от тях се търси средство и място за реализация. Свързан е с влагане на ресурс на етапа на проектиране на приложението, за да се оцени всеки елемент като какъв да бъде класифициран и съответно разпределен към един или друг слой.

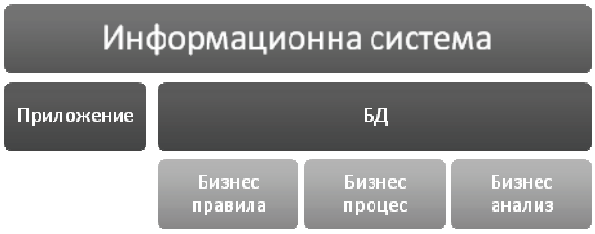
Изхождайки от направения по-горе анализ за същността на алгоритмите и обработките във всеки елемент и подходящите места за реализация при този подход едно потенциално разделяне на бизнес логиката между слоевете на приложението е представено на Фиг. 7.

Тук в максимална степен разработчикът може да се възползва от предимствата и да избегне недостатъците на съхранените процедури.



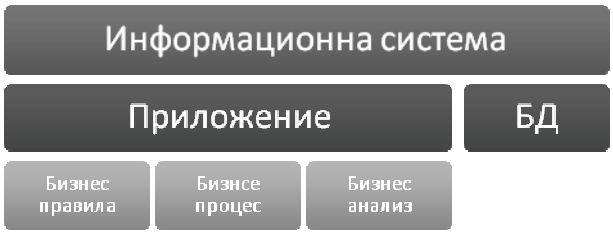
Фиг. 7. Разпределение на бизнес логика при структурен подход

- **Максимален контрол.** Лекотата, с която се поддържат елементите в БД, е голямо изкушение цялата функционалност на едно приложението да бъде реализирана в БД (Фиг. 8). Основни предимства на този подход са:
  – **Лесна администрация**. Промяната в програмните елементи в БД се извършва изцяло прозрачно за потребителя и не изисква подмяна на програмни файлове и компоненти.
  – **Освобождаване от SQL код в приложението**. При този подход програмният код на приложението в най-голяма степен се освобождава от SQL запитвания. Преодолява се немалкото усилие по описание на структурите в БД като обекти в съответния програмен език и дублирането на иначе естествената за БД функционалност по поддръжка на данните. Заедно с това той дава повече възможности за реализация на бизнес логика управлявана от данни (data drive).



Фиг. 8. Разпределение при "Максимален контрол"

  – **БД става най-тясното място.** При този подход обаче БД става най-тясното място в системата, най-критичното, и при използването на съхранените процедури най-много от техните недостатъци могат да се окажат сериозна пречка в поддръжката на бизнес логиката.
  – **БД като неинтелигентна система.** Този подход е отговорът на противниците на съхранените процедури и разглежда БД само като място за съхранение на данните. "Интелектът" на системата е съсредоточен в приложението (Фиг. 9).



Фиг. 9. Разпределение на бизнес логика при БД като неинтелигентна система

- При този подход **се дублира естествената за БД функционалност по поддръжка на данните** в приложението и се влагат много усилия за създаване на структури описващи данните (таблиците) от БД в приложението.
- **Приложението се натоварва с много SQL код**, който трудно се поддържа и оптимизира без това да нарушава работата на потребителя.
- **Използват се неподходящи среди и езици за програмиране за сложни обработки на данни**.
- **Запазва се концепцията на трислойната архитектура, както и теоретичната възможност за смяна на БД**.

- **Управленско-ресурсен подход.** Този подход в практиката се указва най-често прилагания. Преценката, "кое къде да се реализира" зависи от фактори като време за разработка, наличен проектантски и програмистки потенциал. С цел да се реши ad-hoc даден проблем се избира най-бързото и лесно решение, което обикновено е "кой може да го направи най-бързо и с какви ресурси разполагаме в момента".

2.3.2. Класифициране на бизнес логика

Представените по-горе подходи за разделяне на елементите на бизнес логиката отразяват в голяма степен практиката и новия поглед към структурата на БЛ. Всеки от тях има слабости в практическата реализация, свързани с трите основни аспекта от поддръжката на приложението (виж т. 4). Търсенето на възможности за преодоляването им изисква анализ на същността на бизнес логиката. За тази цел следващите анализи и предложения стъпват на идеята на структурния подход при разпределението на БЛ.

Търсейки решение на проблема коя част от бизнес логиката къде да бъде реализирана, е необходимо да са направи анализ и на това: за реализацията на всеки елемент от структурата на БЛ какъв тип програмен код предполага да бъде използван.

Както стана ясно по-горе, програмният код може да бъде разделен на:

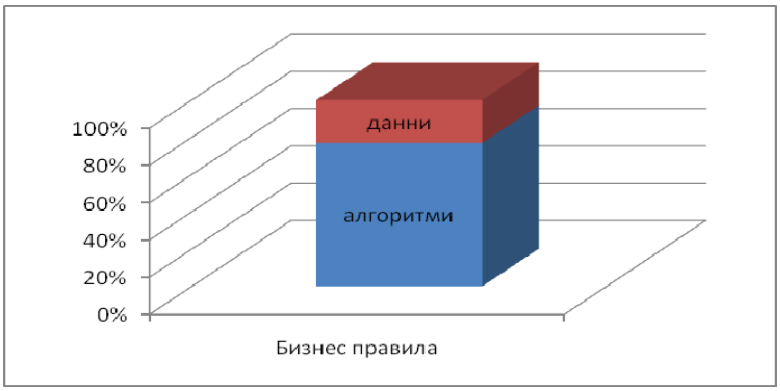
- Свързан с манипулиране на данните;
- Свързан с алгоритмизацията.

Всеки от тези типове има и своя типична и най-присъща реализация:

- Първият тип програмен код най-често се реализира с естествения за обработка на данни език SQL;
- Вторият тип – с помощта на избран език за програмиране.

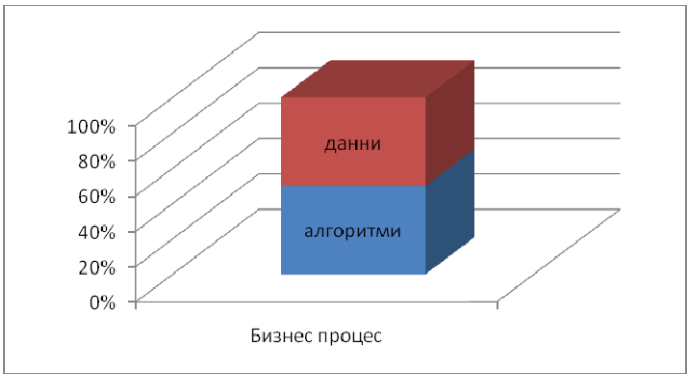
Всеки от представените елементи на БЛ се реализира с различно съотношение на двата типа програмен код.

Бизнес правилата се реализират с превес на алгоритмизацията и по-слабата манипулация на данни (Фиг. 10). Манипулирането на данни в тази част от БЛ има силно транзакционен характер, т.е. работи се с малко данни, обикновено конкретни обекти (редове от същности), и се извършват основни операции по добавяне, редакция и изтриване. Алгоритмите използват извлечени стойности и генерират нови, които се използват в коригиращите операции.



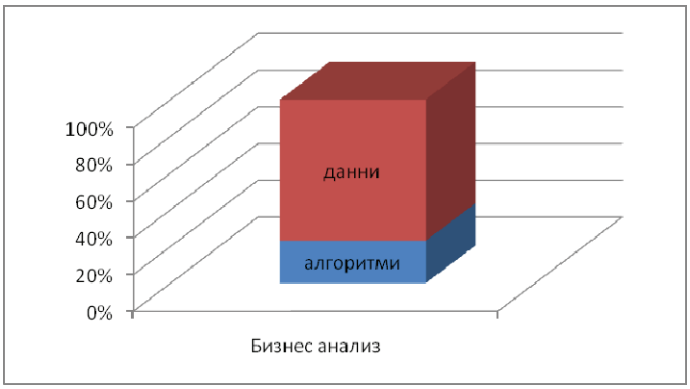
Фиг. 10. Структура на програмен код на бизнес правила

Бизнес процесът изисква както представяне на формални алгоритми, така и е свързан с манипулиране на данни (Фиг. 11). Манипулирането на данни е свързано с работа върху повече от един екземпляр на данните на една същност, а алгоритмизацията е в свързана с формализиране на правила за селектиране на данните и генериране на нови, най-често агрегатни стойности.



Фиг. 11. Структура на програмен код на бизнес процес

Бизнес анализът е компонент на БЛ, който е свързан с манипулиране на по-големи обеми данни и по-слабо представена алгоритмизация (Фиг. 12). Тук манипулирането на данните винаги е свързано с работа върху много редове от много същности и алгоритмите са част от начина на тяхното извличане и особено много са свързани с агрегирането на нови аналитични стойности.



Фиг. 12. Структура на програмен код на бизнес анализ

Разглеждайки по-детайлно структурата на бизнес логиката и най-общо потенциалната реализация на всеки елемент по отношение на вида на програмния код, който се използва, може да се направи опит за класификация на бизнес логиката на едно приложение.

Подобна класификация е необходима, защото чрез нея ще могат да бъдат формулирани и по-конкретни и формални критерии за избор на място за реализация на всеки елемент от БЛ.

На основата на направения анализ може да се предложи следната класификация за бизнес логика:

- Транзакционна бизнес логика;
- Аналитична бизнес логика.

2.3.2.1. Транзакционна бизнес логика

Определение на понятието транзакционна бизнес логика, което се предлага в настоящата студия, е:

Транзакционната бизнес логика е тази част от приложението, която:

- има силно оперативен характер,
- свързана е с реализацията на алгоритми по памет, т.е. обработки, които не изискат работа с БД,
- Манипулирането на данните е свързано с обработване на един екземпляр (ред) на една същност и генерирането на нови, неаналитични стойности за създаване или коригиране на единствен екземпляр (ред).

2.3.2.2. Аналитична бизнес логика

Определение на понятието аналитична бизнес логика, което се предлага в настоящата студия, е:

Аналитичната бизнес логика е тази част от приложението, която:

- не е оперативна,
- е свързаната с реализация на алгоритми по данни, т.е. обработки, силно обвързани с извличане на по-големи обеми от данни от БД, които са повече от един екземпляри (редове) на една същност, и/или на много същности,
- Генерирането на нови стойности от извлечените данни е свързано с произвеждането на агрегатни стойности, не свързани с конкретен ред или същност.

За този вид бизнес логика може да се каже, че в съвременните приложения тя много често има и оперативен, и аналитичен характер. Тази последна особеност е голямото предизвикателство пред избора на място и средства за реализация.

2.3.3. Критерии за класифициране на бизнес логика

Всяка класификация се нуждае от ясни критерии, които да определят принадлежността на даден елемент към определена класификационна група. Критериите дават шаблон, по който да бъдат разпознати всеки от двата вида логика. За класифицирането на бизнес логиката също е необходим набор от критерии за това.

Тези критерии отразяват основните характеристики на бизнес логиката по отношение на аналитичността и транзакционността, на обема обработвани данни, по отношение на резултатите от обработките, по отношение на начина на обработване на данните, които са залегнали в предложените по-горе определения.

Изхождайки от практиката и от структурата на бизнес логиката в настоящата разработка предлагаме следният набор от критерии:

- **Обем на обработвани данни.** Този критерий определя дали се работи с конкретни екземпляри на една същност, или с множество от нейни редове.
- **Обхват на обработваните данни**. Този критерии определя дали се обработват данни от една или повече същности[1].
- **Генериране на аналитични стойности**. Този критерий определя дали резултатите от алгоритмите водят до генериране на нови стойности и дали те са свързани със създаване на нови екземпляри, или имат аналитичен характер.
- **Вид на програмния код**. Този критерий касае избора на това дали се използва предефиниран програмен код, т.е. статичен, присъстващ в реализацията на приложението в пряк вид, и който се компилира, или динамичен, който се създава динамично, по време на изпълнение, при определени ситуации и по набор от правила.
- **Степен на структурираност на предметната област**. Степента на структурираност на предметната област, в която се изгражда едно приложение, е критерии, който е еклектичен и не е свързан с конкретен начин на създаване на приложения. Той присъства в тази класификация, защото отговаря на въпроса "кол-

[1] Под същност се има предвид логическа същност, която може да бъде представена чрез няколко други в конкретната реализация на БД.

ко известни са описанията на данните и алгоритмите за тяхната обработка", а това предопределя колко затворена трябва да бъде реализираната бизнес логика.

Силноструктурираните предметни области се отличават с голямата яснота по-отношение на данните и обработките. За тях може да се разработи изчерпателен програмен код, който да обхваща в максимална степен правилата и ситуациите, а необходимостта от допълване и коригиране е сравнително ниска.

Докато в **слабоструктурираните** предметни области, в които описанията на данните и обработките върху тях са динамични (бързо се променят), създаването на изчерпателен програмен модел е невъзможно и нежелателно. Необходимо е да се предостави възможност за бързо добавяне на нови и за коригиране на съществуващи правила и описания.

За класифицирането на БЛ по предложените критерии не е необходимо едновременно наличие на всеки от тях. Взимането на решение става на основата на мнозинството от стойности на критериите, които го отнасят към конкретна група.

2.3.3.1.Транзакционна бизнес логика

Елемент от бизнес логиката на едно приложение е транзакционната бизнес логика, когато:

- Обемът на данни е 1 ред (екземпляр),
- Обхватът на данните е от една същност,
- Като резултати от приложените алгоритми се генерират нови стойности за създаване на нов екземпляр,
- Програмният код е статичен,
- Предметната област е силностуктурирана.

Или транзакционната бизнес логика е силно обвързана с конкретната същност, която поддържа. *Тя може да се определи като присъщо поведение на конкретна същност, т.е. тя не може да бъде използвана изолирано от тази същност в друг контекст.* Многократното й използване е възможно само в случаите, когато съществува съответната същност.

Транзакционната бизнес логика обхваща бизнес правилата от структурата на БЛ и е подходящо да бъде реализирана в средния слой на приложението, като се използва избран език за програмиране.

2.3.3.2. Аналитична бизнес логика

При аналитична бизнес логика:

- Обемът на данните е повече от 1 редове (извличат, представят, трансформират и т.н.),
- Обхватът на данните е повече от една същност,
- Като резултати от приложените алгоритми се генерират аналитични стойности,
- Програмният код е статичен,
- Предметната област е силностуктурирана.
- Аналитичната бизнес логика обхваща бизнес процесите и бизнес анализите.

Аналитичната бизнес логика не е присъщо поведение на една същност. Този вид логика може да съществува самостоятелно, отделно от данните. *Т.е. за аналитична бизнес логика може да се каже, че има собствен живот и тя може да бъде използвана многократно в друг контекст и дори от друго приложение.* Това поставя проблемът с нейната реализация или по-точно с избора на "място" в архитектурата на приложението.

Обобщение на видовете бизнес логика и тяхното идентифициране според предложения набор от критерии е представено на Фиг. 13.

Критерии / Логика		Обем (Екземпляри)	Обхват (Същност)	Резултатни стойности	Програмен код	Структурираност на предметна област
Транзакционна	**ИЛИ**	1	1	Нови	Статичен	Силно
Аналитична		М	М	Аналитични	Динамичен	Слабо

Фиг. 13. Критерии за класифициране и идентифициране на бизнес логика.

Бизнес логика, реализирана и капсулирана в приложението, е конкретна и тясно обвързана с конкретния контекст. Тя не може да бъде използвана като готов компонент в други случаи. Целта на едно приложение е да предоставя определена функционалност на клиента под формата на интерфейс и не е присъща задачата да предоставя себе си на други приложения за използване.

Разделянето на бизнес логиката би позволило елементите от нея, които не са обвързани и конкретни по своята същност, да бъдат използвани отново. Или ако на тази част от бизнес логиката се осигури друго място за съществуване и се определят правила, така че тя да бъде предоставяна за използване, ще се даде възможност една сериозна част от кода на приложението да придобие ново качество на използване.

Изхождайки от направения анализ на особеностите на аналитичната бизнес логика за нейната реализация, е подходящо да се търси място извън приложението, т.е. подходящо е тя да бъде реализирана така, че да съществува възможност за нейното бързо поддържане и настройване и за многократното й използване. Аналитичната бизнес логика, както стана ясно, е свързана с извличане и обработване на много данни. Това определя базата данни като подходящо място за реализация на този вид бизнес логика.

2.4. Изводи

Като обобщение на направените по-горе анализ и предложения могат да се систематизират следните изводи:

1. Бизнес логиката в едно приложение има своя вътрешна структура. Тази структура може да бъде определена като независима от предметната област и конкретния решаван проблем. Основните елементи на бизнес логиката могат да се формулират като бизнес правила, бизнес процес, бизнес анализ и системна абстракция.

2. Четирите елемента на БЛ присъстват в различно съотношение и могат да бъдат открити, като се разгледат елементите на бизнес логиката, реализирани в различните слоеве на приложението, от гледна точка на предложените в студията критерии: какъв вид алгоритъм е необходим за реализацията и характера на елемента – транзакционен или аналитичен.
3. На основата на разбирането на вътрешната структура на бизнес логиката, предложена в студията, е направен анализ на възможните алтернативи за реализация на всеки от елементите по отношение на неговото място. В студията са изведени четири различни подхода на разпределение на тези елементи: структурен подход, подход "Максимален контрол", "база данни като неинтелигентни системи", управленско-ресурсен подход.
4. Предложеният структурен подход е оптимален по отношение на разпределението на бизнес логиката – неговата поддръжка и възможност за многократно използване. Този подход се отличава с комбинирането на най-доброто.
5. Бизнес логиката може да бъде класифицирана в две големи групи по предложен набор от критерии. Тези две големи групи са аналитична и транзакционна бизнес логика. Разделянето на бизнес логиката на тези две групи е направено изцяло с цел да се определят ясно елементите, които са подходящи за реализация в базата данни и респ. в приложението. За аналитичната бизнес логика е важно, че тя може да съществува относително самостоятелно и е в по-голяма степен многократно използваема в сравнение с транзакционната.

3. КОНТЕЙНЕР ЗА АНАЛИТИЧНА БИЗНЕС ЛОГИКА

Съвременните приложения с многослойна архитектура имат в голямата си част стандартна реализация и разпределение на задачите между слоевете и то представено на Фиг. 14.

Фиг. 14. Трислойна архитектура на приложение

След направения анализ на структурата на бизнес логиката (виж. 2.2) трислойната архитектура може да се разглежда, като в междинния слой съществуват 2 компонента – транзакционна и аналитична бизнес логика (Фиг. 15).

Фиг. 15. Трислойна архитектура на приложение.
Класификация на Бизнес логика

За тези два компонента беше определено мястото за реализация от гледна точка на това къде може да се осигури по лесна поддръжка, настройка и използване. *Транзакционната бизнес логика е по-удачно да бъде реализирана в междинния слой на приложението, а за аналитичната по-подходящо място за съществуване е базата данни* (Фиг. 16).

Фиг. 16. Трислойна архитектура на приложение.
Разделяне на бизнес логиката на приложението

Пренасянето на реализацията на аналитичната бизнес логика в базата данни позволява да се намери решение на проблема с поддръжката на SQL кода в приложението, неговото настройване и оптимизиране. Но при използването на този програмен код възникват някои ограничения, чието преодоляване също би осигурило много сериозна степен на свобода за разработчика при създаване на приложение и при използване на готови програмни фрагменти извън контекста на конкретното приложение.

При програмна реализация на бизнес логика в БД в съвременните приложения се е наложила практиката за силно обвързване на SQL кода (респ. на аналитичната бизнес логика) с конкретна база данни, схема и т.н. Пренасочването на приложе-

ния, т.е. смяна на базата данни или схемата, върху която работят, е тривиална задача. Това обаче изисква поддържане на версии на съответния програмен код на няколко места, което е трудоемка задача и източник на грешки. От друга страна, за всяка една от тези бази данни или схеми е необходимо да се поддържа пълната версия на програмния код (който е избран да се реализира в нея) и ако за част от него е необходимо да се предоставя като външен компонент (услуга), това трябва да се направи във всяка една от тях. *Споделянето на съществуващ програмен код и обекти между базите данни е възможно и е добра основа за търсене на решение за създаване на програмен код с универсално предназначение.* Повечето съвременни СУБД предоставят инструменти за това. Но то винаги е свързано с цитиране на конкретни имена и отново възниква необходимост за пренасочване.

Проблемът с пренасянето на аналитичната бизнес логика в базата данни има няколко интересни аспекта.

Първият е свързан с избора на програмния код, който да бъде реализиран в БД, и предложените критерии за избор (виж 2.3.3. Критерии за класифициране на бизнес логика) дават възможност за неговото решение.

Вторият е свързан с възможността да се осигури прозрачност на тази реализация. Прозрачност по отношение на мястото на реализация на този код и на данните, върху които да се изпълни. Т.е. дали може да се осигури такава реализация, така че аналитичната бизнес логика да може да се използва без изрично споменаване на местоположението. Така един и същи програмен код, който има характер на многократно използваем, да бъде реализиран и поддържан на едно място, но да се изпълнява върху данни на много различни местоположения. Или програмният код, реализиран в БД, да бъде разпределен между много местоположения, но всеки елемент да съществува само веднъж и да може да се изпълнява върху данни от други местоположения.

Аналитичната бизнес логика, която има отношение към манипулирането на данни, е свързана със структурата на данните, която е проектирана и реализирана в конкретната база данни или схема. Но за разлика от транзакционната логика тя има самостоятелен смисъл и не е обвързана с конкретна същност, което я прави добър кандидат да бъде обособена като самостоятелен компонент или компоненти. *Транзакционната логика е подходящо да бъде капсулирана и разглеждана като част от обекта, който манипулира. Докато аналитичната логика не може да бъде "присвоена" на отделна същност*. Нека приведем за разглеждане един пример, чрез който това на пръв поглед странно твърдение да бъде обяснено. Нека се опитаме да разгледаме поставеният проблем в една класическа бизнес информационна система, напр. системата за следене на поръчки. Както теорията и практиката показват, информационната система съществува не само, за да бъдат събирани данни, а и за да могат те да се използват по подходящ начин. Потребителите на системите нямат нужда от просто съхранени данни. Те имат нужда от данни, представени в адекватен за техните нужди вид. А адекватен включва в себе си и изисквания към съдържанието и към начина представяне.

Събирането и съхраняването на данните в съвременните системи следва или функционалното виждане за организация на технологичния процес, или обектната престава за участниците в него, а най-често двете се комбинират. В конкретния

пример поддържането на данните за едни клиент може да бъде определено като транзакционна логика. Поддържането обикновено касае данните на конкретен клиент и е свързано с операциите по добавяне, изтриване и редакция. Тези операции са елементи на бизнес логика, но те нямат смисъл само по себе си извън същността на клиента. Ако разгледаме обаче един списък с клиенти, който е предназначен да ориентира потребителите и включва информация не само за чистите клиентски данни, а и агрегатни величини, отразяващи работата с тях, тук вече тази бизнес логика е по-обща и не касае отделния клиент. *Това е аналитична бизнес логика, която не е присъща само на клиентите.*

Аналитичната логика може да бъде разглеждана като съвкупност от много предефинирани или динамични програмни фрагменти, които могат да бъдат използвани многократно (особено ако бъдат подходящо проектирани и реализирани, така че да се запази именно този характер на необвързаност с конкретна същност).

Аналитичната бизнес логика може да бъде представена чрез много различни видове обекти в базата данни. Съхранените процедури (stored procedures) са подходящ инструмент за представяне на БЛ, която включва не само манипулация на данни, а и алгоритми за получаване на аналитични стойности. Изгледите (views), от своя страна, са широко използван инструмент за наименуване и скриване на заявки за извличане на данни, дори и за манипулиране на данни, доколкото съвременните СУБД предоставят възможност за създаване на тригери върху подобни обекти. Използването на всеки един вид от тези обекти в база данни има специфичен синтаксис на извикване и извличане на данни.

Как може да се осигури възможност тази част от бизнес логиката (в БД) да съществува навсякъде и да бъде достъпна отвсякъде? Т.е. реализирайки приложението, разработчикът да се ангажира с това само да поиска определен елемент аналитична логика по неговото наименование и да посочи върху коя конкретна база данни да бъде приложен?

От друга страна, възможно ли е за разработчикът да си осигури прозрачност по отношение на вида на обекта т.е. за него да остава скрито дали елементът бизнес логика е реализиран чрез съхранена процедура, чрез пакет от съхранени процедури, чрез изглед и т.н.?

От гледна точка на приложението обръщението към всеки един такъв обект е с цел да се получи една или много стойности.

Случаите, когато приложението има нужда само от изпълнение на някаква обработка, се реализира като съхранена процедура, когато няма реално връщане на стойности, не се нарушава общия поглед към комуникацията приложение – аналитична бизнес логика в БД.

Или може да се направи изводът, че взаимоотношението между приложението и аналитичната бизнес логика е свързано с пренос на данни, представящи някакъв резултат, без значение как реално са получени те (т.е. реално приложението не изпълнява заявки).

Т.е. въпросът, пред който се изправя разработчикът, е дали може да превърне своето приложение в по-общ алгоритъм, който само подрежда парченцата БЛ[1] и не се интересува от детайлите за нейната реализация, като по този начин й позволи тя да съществува произволно и самостоятелно.

Търси си се решение, което да позволи създаването на обособено място за поддържане и управление на елементите аналитична бизнес логика, създаване на хранилище, което да се грижи да "знае" достатъчно за аналитичната бизнес логика и да регламентира правила за намиране на, достъп до, изпълнение на тези елементи аналитична логика и представяне и предоставяне на резултатите от изпълнението на заявките. Тези правила да скриват конкретната реализация и местоположение и да осигуряват единствено място на съществуване на тази аналитична логика.

Технологията Java Naming and Directory Interface подсказва подобна идея. В основата на тази технология са т.нар директорийни услуги. Директорийните услуги осигуряват възможност за управление на съхранението и разпределението на споделена информация. Тази информация може да бъде от различни типове – електронни адреси, телефонни номера, IP адреси, особености на печатащи устройства и т.н. Тези услуги управляват съдържанието на директории (каталози), което съдържание може да реферира хора, местоположения или всеки друг конкретен или абстрактен обект. Всеки елемент в директорията има атрибути като имена, идентификатори и др. Чрез тези атрибути елементите се описват и наборът им е специфичен за вида на елемента. Директорийните услуги са обикновени бази от данни, които осигурят конвенционална функционалност, присъща на повечето релационни бази данни, като търсене и филтриране например [12].

Много съвременни технологии реализират подобна идея, но тя е концентрирана върху програмния код изцяло и не дава възможност за отделяне на видовете бизнес логика с цел намиране на най-подходящото място за реализация на всяка една от тях.

Насочвайки вниманието си към аналитичната бизнес логика, възниква идеята за обособяване в архитектурата на приложението на специално място, наречено "контейнер", в което "живеят" елементите на аналитичната бизнес логика. Както Java Naming and Directory Interface се грижи за т.нар Java Been, така е възможно да се разработи контейнер за аналитична логика, реализирана в базата данни (най-често SQL код, изнесен от приложението).

Примерната архитектура на приложението с контейнер за бизнес логика е представена на Фиг. 17.

[1] Отново се говори за аналитична бизнес логика.

Фиг. 17. Трислойна архитектура на приложение с Контейнер на аналитична бизнес логика.

В тази архитектура се запазва основната концепция на многослойните архитектури, а именно разделяне на задачите по съхранение, обработка и представяне на данните.

Новият момент в нея е в това:

1. Приложението в междинния слой се освобождава от SQL кода, чрез който най-често се представя аналитична логика. В него се реализират изключително и само транзакционните алгоритми.
2. Приложението приема нова роля като "потребител на резултати" от изпълнението на някаква логика и като място за абстрактно описание на технологични[1] последователности.
3. Базата данни запазва своята основна задача да бъде хранилище за данните и да не се натоварва с бизнес правила.
4. За SQL кодът се предоставя инструмент за неговото поддържане извън приложението.
5. SQL кодът се предоставя за многократното използване върху различни бази данни или схеми, като по този начина се изгражда "библиотека" на полезните неща, която е прозрачна за разработчиците.
6. SQL кодът се предоставя за многократното използване извън приложението и в различни контексти.

3.1. Спецификация на контейнер за аналитична бизнес логика

Контейнерът за аналитична бизнес логика е предназначен да поддържа подробно описание на местоположението на известните му елементи бизнес логика (най-често реализирани чрез съхранени процедури, изгледи и т.н.) и да гарантира единствено тяхното съществуване.

В своята работа приложението се обръща към контейнера за определен елемент БЛ. Приложението не се интересува къде и как е реализиран елемента. Контейне-

[1] Технологична последователност, заложена в информационната система.

рът обработва заявката, като се опитва да отговори на въпроса познава ли елемента, който е заявен. Ако той е известен на контейнера, същият има за задача да открие местоположението му и заявката от приложението да се трансформира в адекватен изпълним синтаксис, така че програмният код да се изпълни върху конкретните данни, с които работи приложението.

От друга страна контейнерът е отговорен за това да предостави единен механизъм и при заявка определен елемент аналитична логика да бъде предоставен като услуга, като го направи, без да се налага допълнителен програмен код за това.

Като основни характеристики на контейнера за аналитична бизнес логика могат да се дефинират следните:

- **Поддържане на подробна информация за известните му елементи SQL код.**
  Всеки елемент, който е регистриран в контейнера, се идентифицира с наименованието си и се описва с местоположението си. Местоположението може да бъде с различна структура на описанието според избраната реализация. Задача на специфицирания контейнер е да поддържа тази информация и да представя интерфейс за това.
- **Осигуряване на достъп до SQL код, разположен в различни схеми/бази данни.**
  Всеки елемент може да има произволно местоположение, с цел създаване на логическа и физическа организация на процедурния кода в БД. На основата на описанието на всеки регистриран елемент контейнерът трябва да може да предостави подходящ синтаксис за извикване и изпълнение на съответния елемент. Подходящият синтаксис трябва да отрази местоположението на кода и местоположението на данните.
- **SQL код, съхранен в различна схема/база данни, да се изпълнява върху данни в различни схеми/бази данни.**
  Контейнерът има за задача да предоставя подходящ синтаксис за извикване на тези елементи аналитична бизнес логика, който да отрази и местоположението на данните.
- **Осигуряване на многократна използваемост на един и същ SQL код.**
  Регистрирането на елементи в контейнера трябва да гарантира, че всеки един от тях може да бъде използван от различни приложения[1]. Отделянето на процедурните елементи, на аналитичната бизнес логика от базата данни и абстрахирането им от конкретното местоположение на данните позволява те да бъдат лесно използвани многократно.
- **Предоставянето на SQL код като услуга.**
  Актуален аспект на многократната използваемост на аналитичната бизнес логика е предоставянето им като услуги, използвани от други приложения. Чрез предоставянето им като услуга елементите БЛ могат да бъдат използвани в други контексти, от други приложения (работещи върху аналогични структури от данни. Преодоляването на това ограничение е възможно, ако се приложат други подходи за проектиране на бази данни). Задача на контейнера на аналитична

[1] Доколкото остават идентични структурите от данни.

бизнес логика е да осигури интерфейс за пакетиране на SQL код като услуга и той да е единен за всички.

С използването на контейнер за аналитична бизнес логика в архитектурата на приложението се постига:

- Прозрачност на организацията на процедурния код в БД.
- Абстрахиране на реалното местоположение на SQL кода от мястото му на използване.
- Предоставяне на SQL код за използване независимо от приложението.
- Възможност за разпределяне на процедурната логика в БД между различни схеми/бази данни.
- Осигуряване на единно и единствено място за съществуване на SQL код, респ. лесен контрол на версиите.
- Осигуряване на възможност за многократно използване на SQL код.

3.1.1. Технология на функциониране на контейнера за аналитична бизнес логика

Всеки елемент аналитична бизнес логика, който е кандидат за поддържане от контейнера, т.е. за него има смисъл да се използва многократно и се налага да се поддържа често, се регистрира в контейнера. Регистрацията включва съхраняване на информация за Името, Местоположението, Типа, Предназначението на елемента, Версията на елемента.

При регистрацията на един елемент в контейнера се извършва верификация за реалното съществуване на например съхранена процедура на указаното местоположение. Ако съответния елемент не съществува може да бъде създаден, ако при регистрацията се предостави и необходимия изпълним код.

Регистрацията на един елемент може да се актуализира или изтрива.

Към контейнера могат да бъдат изпращани заявки за изпълнение на даден елемент или за предоставяне на информация за него.

Регистрираните елементи в контейнера могат да бъдат заявявани от приложения. Заявяването на елемент от контейнер е изпращане на заявка с името на елемента. Заявката се обработва от контейнера, което включва проверка за наличие. Ако проверката е положителна, контейнерът формира коректен синтаксис за изпълнение на заявения елемент (например съхранена процедура). Отговорът на направената заявка от приложението може да бъде генерираният синтаксис или резултатът от неговото изпълнение. Типът на очаквания отговор се указва в заявката към контейнера.

При заявки за предоставяне на информация контейнерът връща описание на елемента във формат, определен от неговата реализация.

Комуникацията между приложението и контейнера се извършва по регламент, зададен от реализацията на контейнера. Използването на XML като език за специфициране на формата и съдържанието на заявките и техните отговори е удачен избор.

Използването на регистрираните в контейнера елементи може да се извърши по два начина:

- Приложението използва заявения елемент или резултата от неговото изпълнение, като го интерпретира като неделима част от цялата логика. Технологията за реализация на този подход беше дискутирана по-горе.
- Регистрираният елемент се предоставя от контейнера за използване като услуга и приложението се превръща в място за консолидиране и оркестрация на услуги. В случаите, когато даден елемент е заявен като услуга, контейнерът трябва отново да предостави коректен синтаксис за извикване на услуга, като заявеният елемент е подходящо "опакован" и това се извършва автоматично и прозрачно.

3.2. Потенциална реализация на контейнер за бизнес логика

Контейнерът за аналитична бизнес логика може да бъде реализиран със средствата на базите от данни или чрез използване на друг език и среда за програмиране. Имайки предвид, че той е предназначен да управлява аналитична бизнес логика, за която, както бе направен анализът по-горе, удачно място за реализация е базата данни (Фиг. 18).

Фиг. 18. Контейнер на аналитична бизнес логика, реализиран в БД.

В следващото изложение ще бъде представена основната идея за реализация на такъв контейнер със средствата на базата данни.

Реализацията на контейнер за аналитична бизнес логика изисква спецификация на основните структури от данни, в които ще се съхранява информацията за регистрираните елементи, и спецификация на основната функционалност на контейнера.

3.2.1.1. Основни структури на контейнера

За да може контейнерът да изпълнява основите си задачи, е необходимо той да разполага с достатъчно информация за елементите, които са регистрирани в него. Всеки елемент се описва с данни, които го идентифицират и локализират.

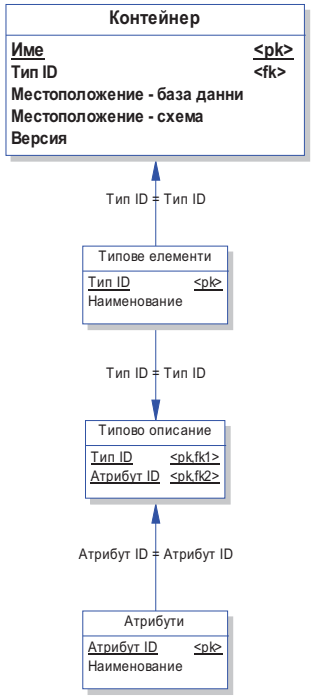
Основните данни за един елемент аналитична бизнес логика са:

- **Име**. Името на елемента може да се задава според правилата за наименуване в контейнера, които са определени от избраната реализация. Името на елемента е неговата идентификация, чрез което формално се скрива конкретното местоположение на елемента. Например в една ORACLE база данни обръщения към обекти (таблици, изгледи, съхранени процедури) от други схеми изисква споменаване на името на схемата. Чрез контейнера се осигурява единствено съществуване на съответния елемент и прозрачност на местоположението, т.е. към контейнера се отправя заявка за елемент с име sp_client, без да се упоменава къде точно е той. Контейнерът чрез името трябва да може да намери достатъчно информация за този елемент, за да осигури използването му.
- **Местоположение**. Местоположението на елемента е съвкупност от няколко атрибута и в зависимост от обхвата на контейнера може да има по-просто или по-сложно описание. Например, ако се разработва контейнер в рамките на един ORACLE Instance, описанието на местоположението ще включва наименованието на схемата, в която съответния елемент е създаден.
- **Тип**. Типът на елемент е важен, тъй като различните типове елементи (съхранени процедури, изгледи, пакетирани съхранени процедури) имат различен синтаксис на използване. Например, ако приложението изпрати заявка за изпълнение на съхранена процедура, контейнерът трябва да успее да подготви подходящ изпълним ред за стартирането й. Ако приложението изпрати заявка за извличане на данни от изглед, контейнерът трябва да успее да генерира съответна select команда върху съответния изглед.
- **Версия на елемента**. Поддържането на версии е един от сложните проблеми в съвременните бази данни. Те не предлагат средства за контрол на версиите. Поддържането на подобна информация в контейнера е част от пълното описание, което трябва да се предоставя при поискване за даден елемент. Поддържането на версия на елемента е възможност и за бъдещо доразвиване на контейнера с инструменти и автоматизирани процедури за контрол на версиите.
- **Описание на елемента**. Описанието на елемента може да бъде разглеждано като набор от много и разнообразни атрибути, които дават допълнителна информация за елемента, но нямат пряко значение за локализацията и стартирането на елемента. Този набор зависи от типа на елемента и би могъл да бъде произволно разширяем.

Един примерен набор от структури, в които да бъдат описвани елементите на контейнера, е представен на Фиг. 19.

Предложената структура е обща и тя има за цел да разкрие необходимите същности. Тяхното прецизиране е предмет на конкретна реализация на идеята.

В предложения концептуален проект същности “Атрибути”, “Типово описание” и “Типове елементи” са част от реализацията на идеята описанието на елементите да бъде зависимо от техния тип и в същото време да следва определен шаблон за този тип, с възможност то да се допълва и разширява [4]. Същността “Атрибути” може да бъде разглеждана като речник на атрибути, в които могат да се описват различни обекти. Този речник е разширяем, т.е. в него могат да се добавят нови редове, представящи нови характеристики.



Фиг. 19. Структура на контейнер на аналитична бизнес логика, реализиран в БД.[1]

Същността "Типове елементи" е номенклатурата на типовете елементи бизнес логика, които контейнера познава. Разширяването на познатите типове изисква разширяване и на реализацията на контейнера, така че за всеки нов тип да бъдат ясно специфицирани правилата за генериране на синтаксис.

"Типовото описание" е същността, чрез която се представя шаблонът, по който се описва един елемент аналитична бизнес логика от даден тип.

Основната същност за контейнера е едноименната, която има фиксиран набор от атрибути, касаещи идентификацията и локализацията на елемента.

3.2.1.2. Основни функционалности и методи на контейнера

Контейнерът за аналитична бизнес логика обхваща функционалност, която може да бъде разделена в няколко основни групи:

- Управление на съдържанието
- Изпълнение на заявки
- Предоставяне на услуги

Тези основни групи съдържат множество от методи, които реализират конкретни задачи.

[1] pk – primary key; fk_n – foreign key.

3.2.1.2.1. Управление на съдържание

Управлението на съдържанието включва предоставяне на програмен интерфейс за поддържане на основната структура от данни на контейнера.

Основните методи в тази група са:

- ***Регистриране на елемент***
  Регистрацията на елемент в контейнер може да се изпълнява с или без верификация за съществуване и със или без създаване на елемента. При регистрация на елемент в контейнер се попълват основните данни в предвидените структури и изпълняване на процедури, за проверка и създаване. Тези процедури са единни и общи за всички типове елементи (изгледи, съхранени процедури и т.н.).
- ***Изтриване на елемент***
  Изтриването на елемент от контейнер заличава данните за елемента, но не премахва самия елемент от неговото местоположение. След тази операция елементът вече не е познат за контейнера.
- ***Актуализация на елемент***
  Актуализацията на елемент в контейнер включва промяна на основните му данни и/или създаване на нова версия на елемента на оригиналното му местоположение или коригиране на съществуваща версия.
- ***Синхронизиране на контейнер***
  Всеки от регистрираните в контейнера елементи съществува на определеното местоположение и може да бъде манипулиран независимо от контейнера. Това може да доведе до разминаване в описанието на елемента и реалното му съществуване. Синхронизирането на контейнера е процедура, чрез която се извършват проверки за всеки регистриран елемент и евентуално актуализиране на данни за него.
- ***Търсене на елемент***
  Работата с контейнера може да е свързана само с преглеждане на неговото съдържание и предоставяне на информация за един или повече елементи. Търсенето на елемент е свързано със задаване на критерии за откриване на елементи и предоставяне на тяхното описание. Предоставянето на описанието всъщност е вътрешна заявка към друг метод в контейнера, който е натоварен с тази задача.

3.2.1.2.2. Изпълнение на заявки

Изпълнението на заявки е свързано с предоставяне на единен интерфейс за обработка на заявките към контейнера и връщането на резултата в заявен от приложението вид. Изпълнението на заявки в повечето случаи е свързано с предаване на параметри към конкретен елемент бизнес логика (като съхранени процедури) и с предаване на резултат от изпълнението към приложението. Тази комуникация се извършва чрез предаване на съобщения XML формат, който всеки контейнер може да специфицира сам.

Заявката може да бъде за извличане на данни от изгледи, за изпълнение на съхранени процедури и др. Изискването на контейнера е те да бъдат наименовани и по това име да се разпознават. Обвързаността с конкретните структури в базата данни

се преодолява по различни начини и те са в пряка връзка с конкретната реализация, която ще бъде избрана.

Основните методи в тази група са:

- ***Изпълнение на заявки с връщане на синтаксис***
  Както беше анализирано по-горе, обобщеното взаимоотношение между приложението и базата данни е получаване на резултат при изпълнение на заявка, който резултат може да има различен вид.

  Конвенционалната комуникация между приложенията и базите данни е свързана с изпращане от страна на приложението на конкретни синтаксиси за изпълнение към базата данни, която от своя страна връща резултата от изпълнението му.

  При работа с контейнер може да се запази тази технология на работа, като контейнерът се използва само като "преводач", който конкретизира една обща заявка в много конкретна.

  Предоставянето на синтаксиса има за цел да върне към приложението не резултат от изпълнение, а програмния код за това. Тази възможност е необходима, тъй като често приложенията взимат решение за изпълняване на една или друга функционалност, като се анализират заявките, които реално се изпълняват.

  Този синтаксис по-късно може да бъде изпълнен от приложението, като се използват стандартните технологии.
- ***Изпълнение на заявки с връщане на резултат***
  Този метод всъщност е изразител на идеята за прозрачност и на разбирането за общата форма на комуникация между приложението и базата данни.

  Чрез този метод контейнерът е отговорен не само да подготви съответния синтаксис, но и да го изпълни и резултатът от изпълнението да бъде върнат към приложението.
- ***Изпълнение на заявки с връщане на описание***
  Всеки регистриран елемент в контейнера има описание, което, както бе предложено по-горе, има задължително статично описание, което го определя еднозначно, и набор от атрибути, които го специфицират по-точно. Методът предоставя на приложението описанието на елемента бизнес логика, отново с цел то да извърши някакви допълнителни анализи и разпознаване на елемент, необходими му за изпълнение на неговата функционалност. Този метод може да бъде използван и извън приложения, в комбинация с методите за търсене на елементи.

3.2.1.2.3. Предоставяне на услуги

Основните методи в тази група са:

- ***Предоставяне на елемент като услуга***
  Всеки регистриран елемент в контейнера може да бъде използван по два начина: като логическа част от приложението или като самостоятелен обект, който е достатъчно завършен и може да бъде добавян към други функционалности. Първият аспект на използване се реализира чрез методи на контейнера от група "Изпълнение на заявки". Вторият начин за използване е чрез предоставяне на елементите аналитична бизнес логика като услуги, които да бъда използвани

като такива. Контейнерът е отговорен да извърши необходимите действия (синтактични най-вече), за да представи елемента като.

3.3. Съхранени процедури за примерна реализация на контейнер за бизнес логика

Основните методи на контейнера по същество са алгоритми, които използват и/или поддържат структурите от данни в контейнера. Следователно те имат силно процедурен характер и за тях е подходяща съответна процедурна реализация.

В съвременната база от данни процедурните аспекти имат своята най-добра реализация като съхранени процедури.

Съхранените процедури комбинират в себе си синтаксиса и изпълнението на процедурен език и език за манипулиране на данни (SQL), което ги прави достатъчно добър инструмент за реализация на предложената функционалност на контейнера.

Изборът на този инструмент е подпомогнат и от факта, че предложеното решение за контейнер на аналитична бизнес логика се прави в търсенето на отговор на въпроса бизнес логиката може ли да съществува извън приложението и дали базата от данни е добро място за това.

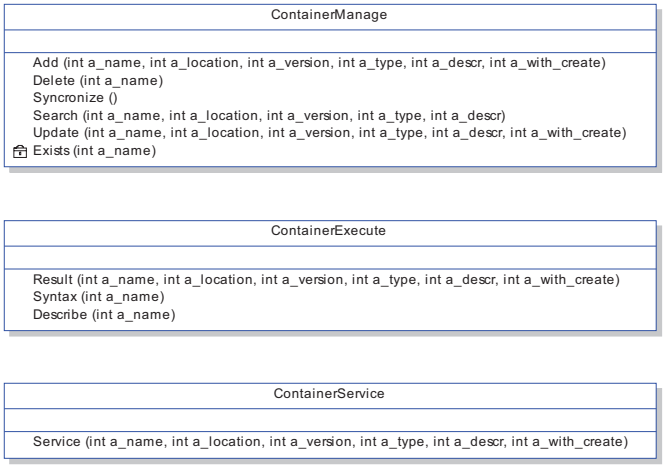
Примерната реализация ще бъде представена като общ скелет на пакетите от процедури, както и като общ вид на потенциалните алгоритми. Тази реализация е ограничена и стъпва на предпоставката, че елементите бизнес логика са разпределени между различни схеми на една и съща база данни[1].

Конкретната реализация изисква много по-подробен програмен код и не изключва комбиниране на процедурните възможности на базите данни с използване на други езици за програмиране с цел постигане на бързодействие, добра организация на програмния код и др.

Основните методи на контейнера е подходящо да бъдат обособени в пакети, като всеки пакет обхваща методите от определена група. Всеки метод се представя като съхранена процедура. В рамките на всеки пакет основните методи са публични т.е. достъпни извън него. За нуждите на функционалността на всеки метод пакетите съдържат скрити процедури и функции, които имат за цел да обособят отделни части от програмния код като самостоятелни, многократно използваеми елементи. Предложение за организация на набора от процедури и функции е представено на Фиг. 20.

В следващите точки ще бъде предложена примерна реализация на съхранени процедури за реализация на основни методи на контейнери и техни прототипи, реализирани с PL/SQL. Тези реализации са основните прототипи, на които биха могли да бъдат дадени различни дефиниции, според вижданията на разработчиците за ефективност и правилна организация на програмен код.

[1] Използва се концепцията за организация на схемите в СУБД Oracle, както и част от синтаксиса на езика PL/SQL.



Фиг. 20. Методи на контейнер на аналитична бизнес логика, реализиран в БД

3.3.1. Управление на съдържание

```
Create or replace package ContainerManage
Begin
--процедура за създаване на елемент
        Procedure Create (a_source,a_location);
--процедура за добавяне на елемент
        Procedure Add (a_name,a_location,a_version,a_type,a_descr,a_create);
--процедура за изтриване на елемент
        Procedure Delete (a_name);
--процедура за актуализация на елемент
        Procedure Update
     (a_name,a_location,a_version,a_type,a_descr,a_create);
--процедура за синхронизиране на контейнер
        Procedure Synchronize;
--процедура за търсене на елементи
        Procedure Search (a_name,a_location,a_version,a_type,a_descr);
--функция за проверка за съществуване на елемент
        Function Exists(a_name)  return number;

        not_exisist_element exception;

End;

Create or replace package body ContainerManage
Begin
        Procedure Create (a_source, a_location)
        Begin
            --връзка към указаното местоположение
               Execute immediate "connect….";
```

```
        --изпълнение на предоставения код за създаване на елемента
            Execute immediate a_source;
        --прекъсване на връзката към местоположението на елемента
            Execute immediate "disconnect…";
        --връщане към контейнера
            Execute immediate "connect….";
    End Create;

    Procedure Add (a_name,a_location,a_version,a_type,a_descr,a_create)
    Is

    duplicate_element exception;

    begin
    --ако не елементът съществува, се регистрира
    --в противен случай контейнерът връща грешка

            If not Exists(a_name)then

            Добавяне на запис в основните структури на контейнера;
            --ако заявката за регистриране на елемент със създаване на
        елемента
            --се извършва съответното действие
            If a_with_create = 1 then
                    Create(a_source, a_location);
            End if;
        Else
                    Raise duplicate_element;
            End if;

    End add;

    Procedure Update (a_name,a_location,a_version,a_type,a_descr,a_create)
    Is

    not_exisist_element exception;

    begin
    --ако елементът съществува, се актуализира
    --в противен случай контейнерът връща грешка

            If Exists(a_name)then

            Актуализация на основните структури на контейнера;
            --ако заявката за актуализация на елемент със създаване на
        елемента
            --се извършва съответното действие
            If a_with_create = 1 then
                    Create(a_source, a_location);
            End if;
        Else
                    Raise not_exisist_element;
            End if;
```

```
        End Update;

        Procedure Delete (a_name)
        Is

        begin
        --ако елементът съществува, се изтрива
        --в противен случай контейнерът връща грешка
                If Exists(a_name)then
                    Изтриване на основните структури на контейнера;
             Else
                     Raise not_exisist_element;
                End if;

        End Delete;

        Procedure Search (a_name,a_location,a_version,a_type,a_descr)

        Is

        begin
          --създаване на динамични заявки по зададените критерии
          Генериране на заявка и записване в променлива select_statment;
          Execute immediate “insert into tmp_resutl ;”||select_statement;
          --възможна реализация на търсенето е резултатът от динамичната заявка да се съхрани
          -- във временна таблица или в таблица
          --с подходяща индентификация на собственика на резултата

        End Search;
        Function Exists (a_name) return number
        Is
        In_scheme number;
        begin
                --проверка за съществуване на елемент в някоя схема на базата данни
                --например чрез използване на системни таблици
                     Select count(*)
                             into in_scheme
                                 from all_objects where name = a_name;

                return in_scheme;

        End Exists;
End;
```

3.3.2. Изпълнение на заявки

```
Create or replace package ContainerExecute
Begin
    --процедура за получаване на резултат от изпълнение на заявката
        Procedure Result (a_name,a_param_xml,a_result);
    --процедура за получаване на синтаксис за изпълнение на заявката
        Procedure Syntax (a_name, a_param_xml,a_syntax);
    --процедура за получаване на описание не елемент
        Procedure Describe (a_name, a_description);

        not_exisist_element exception;

End ;

Create or replace package body ContainerExecute

Procedure Result (a_name,a_param_xml,a_result)
Is
        A_syntax varchar2(…);
begin
    If ContainerManage.Exists = 1 then
    --ако елементът съществува, се изготвя подходящия синтаксис
            Syntax (a_name, a_syntax);
    --изготвеният синтаксис се изпълнява
            Execute immediate a_syntax;
    --подготвя се резултат за връщане
            A_result = …
    Else
        --в противен случай контейнерът връща грешка
            Raise not_exisist_element;
    End if;

End result;

Procedure Syntax (a_name,a_param_xml,a_syntax)
Is
Begin
        If ContainerManage.Exists = 1 then
            --ако елементът съществува се изготвя подходящия синтаксис
            Изготвя се коректният синтаксис;

            --подготвя се синтаксис за връщане
            A_syntax =…

        Else
        --в противен случай контейнерът връща грешка
```

```
            Raise not_exisist_element;
        End if;
End Syntax;

Procedure Describe (a_name, a_description)
Is
Begin
        If ContainerManage.Exists = 1 then
            --ако елементът съществува, се извлича описанието му
                Извлича се описанието на елемента

            --подготвя се описание за връщане
                A_descrition = …

        Else
        --в противен случай контейнерът връща грешка
            Raise not_exisist_element;
        End if;
End Describe;

End;
```

3.3.3. Предоставяне на услуги

```
Create or replace package ContainerService
Begin
    --процедура за предоставяне на елемент като услуга
    Procedure Service (a_name);
End;

Create or replace package body ContainerService
Begin
Procedure Service (a_name)
Is
Begin
        If ContainerManage.Exists = 1 then
            --ако елементът съществува, се изпълняват необходими методи
    (извикване на процедури)
            -- за обявяването му като услуга
                Изпълнение на процедури за обявяване като услуги [9]
        Else
            --в противен случай контейнерът връща грешка
            Raise not_exisist_element;
        End if;

End Service;
End;
```

Важен аспект при реализацията на контейнера е преодоляването на зависимостта на аналитичната бизнес логика от конкретните структури от данни (в смисъл на техните наименования). При предложената тук реализация в рамките на различни схеми в една база данни ORACLE може да се използват:

- Възможностите за създаване на обекти, независимо че ще бъдат създадени невалидни (FORCE опция);
- Възможностите за изпълняване на динамичен SQL (чрез execute immediate);
- Чрез съхранение на елементите бизнес логика в некомпилиран вид и компилирането им при заявка в съответна схема (създаване на локални копия) и унищожаването им след получаването на резултата.

Задълбоченото дискутиране на тези детайли е извън целта на настоящата разработка и е предмет на друг по-обстоен анализ.

Използването на контейнера в кода на приложенията се свежда до това местата, в които се налага използване на SQL код или друг елемент бизнес логика, да се заместват с извикване на съответна процедура от съответен пакет. Така в приложението остава само маркер за мястото на потенциалния SQL код. Извикването на процедурата скрива този код и заедно с това освобождава разработчика от необходимостта да цитира местоположения и т.н. Един кратък пример: извличането на списък на клиенти с допълнителни агрегатни данни за техните поръчки се извършва чрез изпълнение на създадена за целта съхранена процедура sp_client_list, която генерира съответния резултат. В конвенционалната реализация в кода на приложението би било записано select * from client. В новата архитектура там би било записано **exec ContainerExecute.Result('client_list').** И тук като резултат от изпълнението приложението ще получи точен набор данни, например в XML формат. Друг вариант е извикването **exec ContainerExecute.Syntax("client_list");** и като резултат приложението, отново в XML формат, ще получи точният синтаксис за изпълнение на процедурата, създаваща този списък клиенти exec sp_client_list. Предаването на параметри се организира, чрез XML, който е достатъчно общ формат, за да могат да се опишат разнообразни параметри.

3.4. Изводи

Като обобщение на направените по-горе анализи и на предложената конкретна реализация на основната идея на автора на студията, могат да се изложат следните изводи:

1. На основата на разделянето на бизнес логиката и даването на ясни критерии за това, многослойната архитектура на приложението може да се допълни с нов слой с абстрактно предназначение – контейнер за аналитична бизнес логика. Чрез него приложението се абстрахира в максимална степен от аналитичната бизнес логика. Това позволява много лесна поддръжка и постигане на висока степен на прозрачност при работата с данните.
2. Чрез предложения контейнер за аналитична бизнес логика се псотига по-висока степен на многократно използване на програмен код, реализиран в базата данни, който е абстрахиран и от конкретното си местоположение.

3. Чрез предложения контейнер за аналитична бизнес логика се дава възможност да се разработи общ механизъм за извеждане на програмен код от базата данни на по-високо ниво – като услуги, достъпни и приложими върху различни данни.

4. ЗАКЛЮЧЕНИЕ

В последните години разработването на бизнес приложения получи нов облик. Традиционното проектиране на информационни системи с фиксирана, предефинирана функционалност все по-често се сблъсква с проблеми, свързани с настройването и оптимизирането. Информационните системи придобиват по-различен облик. Те се превръщат от точен превод на бизнес процесите в място за комбиниране и подреждане на бизнес логика. Какво е по-различното?

Създаването на многобройните системи в областта на бизнеса и различните варианти на формализация на процесите, които те пораждат, доведоха до създаване на много и различни програмни модели на бизнес обекти и процеси с различно ниво на абстракция. Необходимостта от бързо създаване на бизнес приложения роди идеите за многократното използване на различни обекти, които в началото бяха абстрактни и несвързани с конкретна предметна област, до сегашните технологични решения – семантично натоварени обекти, които да могат да бъдат използвани многократно. По този начин приложенията в една или друга степен се превръщат в място, в което се подреждат различни готови елементи, за които се очаква, че реализират някаква необходима функционалност.

Съществуват много технологични решения, които позволяват това събиране и подреждане на парченцата на пъзела да се случва.

Предизвикателството, на което отговориха съвременните технологии беше е предоставянето на възможност елементи от приложенията да бъдат обособявани като отделни, многократно използваеми елементи. Освен че откриха тази възможност пред разработчиците, те породиха нов проблем. Този нов проблем възникна от алтернативите за разработка на тези използваеми елементи, пред които разработчиците се изправиха. Еднакво добри кандидати се оказаха технологии, използвани в различни нива на архитектурата на приложението. В практиката се наложи един по-общ поглед към функционалността на приложението като бизнес логика, чието място за реализация е ясно определено.

Този общ подход създаде предпоставки за търсене на нови технологични решения, които да позволят добрата идея за многократното използване да бъде приложена в по-широк спектър програмни елементи, като при това тези елементи са намерили най-доброто място за своето съществуване.

В настоящата разработка се прави опит да се предложи по-задълбочен поглед върху бизнес логиката на приложението и по-ясни определения за нейната семантика и структура.

Направеният анализ показва, че наложилият се в практиката подход на съсредоточаване на бизнес логиката в средния слой на приложението, създава проблеми с лесната поддръжка и оптимизация на приложенията.

Разглеждането на бизнес логиката като съвкупност от няколко структурни еле-

мента и дефинирането на ясни правила за класифициране на конкретния програмен код дават възможност бизнес логиката да бъде разпределена между средния слой и сървъра в архитектурата.

Подобно разпределение не е произволно и позволява всеки програмен елемент да се изпълнява на най-подходящото за него място (т.е. като се използва възможно най-подходящата среда за това), като се запазват възможностите за оптимизация и настройка на приложението. По този начин не се променя архитектурата на приложението, но разработчиците могат да се възползват от добрите идеи в нея изцяло.

Пренасянето на бизнес логиката в сървъра (конкретно в базите данни) дава повече свобода при разработката на приложението. Но тя не е достатъчна.

В тази посока беше предложена концепция за изграждане на нов компонент в архитектурата на приложението, който да има задачата да управлява пренесената в БД бизнес логика. Това управление е с цел бизнес логиката в БД да може да се абстрахира максимално от приложението и на него да се предостави само споменатата по-горе задача да бъде "диригент" на "оркестър" от програмни елементи. И заедно с това на всеки от тези елементи бизнес логика да се осигури и самостоятелен живот, т.е. те да могат да се използват извън приложението (в среди и контексти, които позволяват това, доколкото тази логика често остава свързана със структурите от данни).

Този нов компонент, наречен *контейнер за аналитична бизнес логика*, не е създаден самоцелно. Обособяването му като отделен елемент в архитектурата на приложението цели постигане на ефективност в няколко посоки – по отношение на:

- Разработка на приложение;
- Поддържане на приложение;
- Създаване и използване на готови елементи.

Предложената концепция за контейнер на аналитична бизнес логика очертава основните изисквания към подобен компонент и дава някои идеи за конкретна реализация, които са непълни, тъй като целта на настоящата разработка не е реализацията му.

ИЗПОЛЗВАНА ЛИТЕРАТУРА

1. Лазарова В., Е. Денчев, Д. Велев, А. Мурджева, В. Кисимов, К. Стефанова, М. Цанева. Критерии за определяне на потребителската ефективност на информационните системи в Интернет. В: „Бизнес информатика"- сборник доклади от юбилейна международна научна конференция по повод 40-та годишнина на специалност Информатика, С., (октомври, 2007), стр. 223 – 230
2. Мурджева А., Разделяне на бизнес логиката в многослойни приложения. Съхранените процедури като средство за реализация на бизнес логика, доклад, „Бизнес информатика"- сборник доклади от юбилейна международна научна конференция по повод 40-та годишнина на специалност Информатика, (октомври 2007)
3. Мурджева А, К. Стефанова, М. Цанева, В. Лазарова, Е. Денчев, Д. Велев,. Разделяне на бизнес логиката в многослойни приложения и съвременните технологии. В: „Бизнес информатика"- сборник доклади от юбилейна международна научна конференция по повод 40-та годишнина на специалност Информатика, С., (октомври, 2007), стр. 231-242
4. Мурджева А., Сходство и динамика на информационните обекти. Проектиране на единни релационни структури, студия, Научни трудове на УНСС, (1999)
5. Цанева Моника, В. Кисимов, Е. Денчев, А. Мурджева, Д. Велев, К. Стефанова, В. Лазарова. Интегриране на бизнес приложения – основно предизвикателство към системното програмиране. В: „Бизнес информатика"- сборник доклади от юбилейна международна научна конференция по повод 40-та годишнина на специалност Информатика, С., (октомври, 2007), стр.215-222
6. Bouma Frans, Stored procedures are bad, m"kay?, сайт: ASP .NET WebLogs, (ноември, 2003)
7. Chad Z. Hower aka Kudzu, Dude, where"s my business logic, сайт: http://www.woohoo.net (2006)
8. Cunningham & Cunningham, Inc. Business Logic In Stored Procedures, (2005)
9. Developing and Deploying Stored Procedure Web Services, документация на ORACLE 9
10. http://download-west.oracle.com/docs/cd/B14099_19/web.1012/b14027/plsqlservices.htm
11. http://en.wikipedia.org/wiki/Business_logic
12. Sundsted T., JNDI overview, Part 1: An introduction to naming services, Use JNDI to better manage your distributed applications, сайт: http://www.javaworld.com, (януари 2000)

13. Khawar Zaman Ahmed, Cary E. Umrysh, Introduction to Enterprise Software, (2001)
14. King Gavin, Interview with founder of Hibernate, сайт: http://www.javafree.org, (ноември 2002)
15. Mandayam Parthasarathy , SQL Server News: Why use stored procedures? (2002)
16. Marston Tony, Stored Procedures are EVIL, (2006)
17. Matlis Jan,QuickStudy: Business Process Execution Language (BPEL)
18. Miller D. J, Good and Evil in the Garden of Stored Procedures, сайт: http://codebetter.com, (май 2005)

РАЗПРЕДЕЛЕНА БИЗНЕС ЛОГИКА В ИНФОРМАЦИОННИ СИСТЕМИ. ИЗГРАЖДАНЕ НА КОНТЕЙНЕР ЗА АНАЛИТИЧНА БИЗНЕС ЛОГИКА

Резюме

Настоящата студия е изследване на възможности за разделяне на бизнес логиката на информационните системи и на алтернативи за нейната реализация. Предложен е анализ на възможностите за реализация на бизнес логика в информационни системи в многослойна архитектура. В представения анализ се разглеждат съществуващите в практиката алтернативи и се дава оценка на положителните и отрицателните страни на всяка една от тях.

На основата на направения анализ в студията се прави предложение за по-задълбочено разглеждане на понятието бизнес логика, като се акцентира върху структурата на бизнес логиката. Предлага се нова дефиниция и анализ на структурата на бизнес логиката и на възможностите за разпределение на нейните компоненти между слоевете на приложението.

Изхождайки от дефинираната структура на бизнес логиката, се предлага система от критерии за класифициране на елементите на едно приложение като принадлежащи към един или друг компонент на логиката. Използвайки предложените критерии, се дефинират два основни типа бизнес логика, като за всеки от тях се задава стойност на критериите и възможностите за реализация.

В частта за дефинираната аналитична бизнес логика се прави анализ на възможностите за реализация и слабите места на съществуващите подходи по отношение на използваемостта, настройването и поддържането на тази част от програмния код. За преодоляване на тези слабости в студията се представя идеята за допълване на многослойната архитектура с нов елемент, наречен контейнер за аналитична бизнес логика, като се дава спецификация на предназначението му. По отношение на предложения контейнер се прави анализ на подходящото средство за реализация, каквото са съхранените процедури, и се предлага общ скелет на потенциален набор от съхранени процедури.

.

SPLITTING BUSINESS LOGIC. CONTAINER FOR ANALYTICAL BUSINESS LOGIC

Abstract

This study is an examination of the possibilities to split the business logic of information system between the different layers in the application architecture and the existing alternatives to realize the business logic. In the analysis are discussed the alternatives from the point of view of the possibilities, that they give to the developers and users of the system, to reuse, optimize and maintain the business logic.

On the ground of the given analysis in the study is proposed new definition for the term "business logic". This new definition looks at the structure of the business logic and is a very important ground to make more extensive research of the components and their native place for "living", for realization.

To make decision about the better place for the business logic components, are necessary to define a clear set of criteria to classify the elements of information systems as some kind business logic. The study makes some proposition about such criteria set and use them to define two major kinds of business logic – transaction business logic and analytical business logic. For each of them are given values for the criteria, that easy to find.

Further in the paper for one of the business logic kinds, exactly then one called "analytical business" , are made more extensive research. The results of the research are in the direction to explore the weakness of existing technology and approaches to realize this kind of business logic. The points of view here again are optimization, reuse and maintenance. To overcome the weaknesses in the study is introduced a new idea to put a new level, a new component in the application architecture. This new component is called "container for analytical business logic" and his purpose is to manage the analytical business logic so that to ensure the existence of the component in only one place and to give access to different (as purpose, as realization) elements. As argument of the acceptability of this idea in the study also is proposed a concrete instrument to realize the container and as such instrument are chosen stored procedures. To illustrate better the functionality and the use technology of the container is given a set of prototypes of packages and stored procedures as a mainframe as possible and very native instrument for realization of the architecture component.