Факултет инжењерских наука Универзитета у Крагујевцу

Дејан С. Милосављевић

Визуелно моделирање база података у програмском пакету *Rational Rose*

завршни рад

Крагујевац, 2017.

Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма:Машинско инжењерство
Ниво студија: Основне академске студије
Модул: Информатика у инжењерству
Предмет: Базе података
Број индекса: 75/2014

Дејан С. Милосављевић

Визуелно моделирање база података у програмском пакету *Rational Rose*

завршни рад

Комисија за преглед и одбрану:

1. **Др Милан Ерић - ментор**
2. _____
3. _____

Датум одбране:_____

Оцена:_____

Изјава о самосталној изради рада

Изјављујем да сам овај завршни рад урадио самостално, служећи се стеченим знањем и искуством као и наведеном литературом.

75/2014 Дејан С. Милосављевић

_____

(потпис)

У оквиру завршног рада са темом "*Визуелно моделирање база података у програмском пакету Rational Rose***" кандидат треба да:**

Проучи литературу и презентира примену програмског пакета *Rational Rose* у пројектовању релационих база података. Рад треба да обухвати уводна разматрања везана за базе података и њихово пројектовање, примену UML-а, опис елемената UML-а и опис дијаграма које подржава UML, окружење и могућности програмског пакета *Rational Rose*. На конкретном примеру приказати примену програмског пакета. На крају дати закључна разматрања и списак коришћене литературе.

Препоручена литература:

1. Полишчук Ј.: **Пројектовање информационих система**, Електротехнички факултет, Подгорица, 2007.
2. Rainer K., Turban E.: **Introduction to Information Systems**, John Wiley & Sons, Inc., 2009
3. Лазаревић Б., ……: **Базе података**, ФОН Београд, Београд 2003.
4. Вељовић А.: **Основе објектног моделирања - UML**, Компјутер библиотека, 2012.
5. Naiburg E., Maksimchuk R.: **UML for Database Design**, Addison-Wesley, Boston, 2012.
6. Terry Quatrani, **Visual Modeling with Rational Rose 2002 and UML**, Addision-Wesley 2003

Крагујевац, јун 2017.

Ментор:

Prof. Dr Milan D. Erić

Summary

In this paper, it will be explained the emergence and development of databases, that is, what are it's initial beginnings and who is considered it's founder. We will pass through the historical development and demonstrate how the development of databases came about. Explain some databases models, what thay consist of and what these parts do. We will show what the entitets, attributes, keys and relations men, and why thay are important for each database cteation. It will be explained what visual modeling is and which programming language is used this modeling. The visual modeling display will be done in one of the modeling tools called Ratonal Rose, where one of the basic parts and the possibilities of creating various diagrams in this program will be explained. A diagram that will be described in more detals is the Class Diagram along with it's parts and an example done in the aforementioned program.

Key words: databases, relations, keys, diagrams, modeling, UML, Ration Rose, operation, cardinality.

Резиме

У овом раду биће објашњен настанак и развој база података, односно који су њени први почеци и ко се сматра њеним оснивачем. Проћићемо кроз историјски развој и показати како је долазило до развоја база података. Објаснити неке моделе база података, од чега се они састоје и чему служе ти делови. Показаћемо шта значе ентитети, атрибути, кључеви и везе и зашто су они важни при сваком креирању базе података. Биће објашњено шта представља визуално моделирање и који се прогрмски језик користи при овом моделирању. Приказ визуалног моделирања биће одрађен у једном од програма за моделирање која се зове *Rational Rose* где ће бити објашњени неки од основих делова и могућности израде разних дијаграма у овом програму. Дијагам који ће бити детаљније описан је Class Diagram заједно са својим деловима и примером одрађеним у већ поменутом програму.

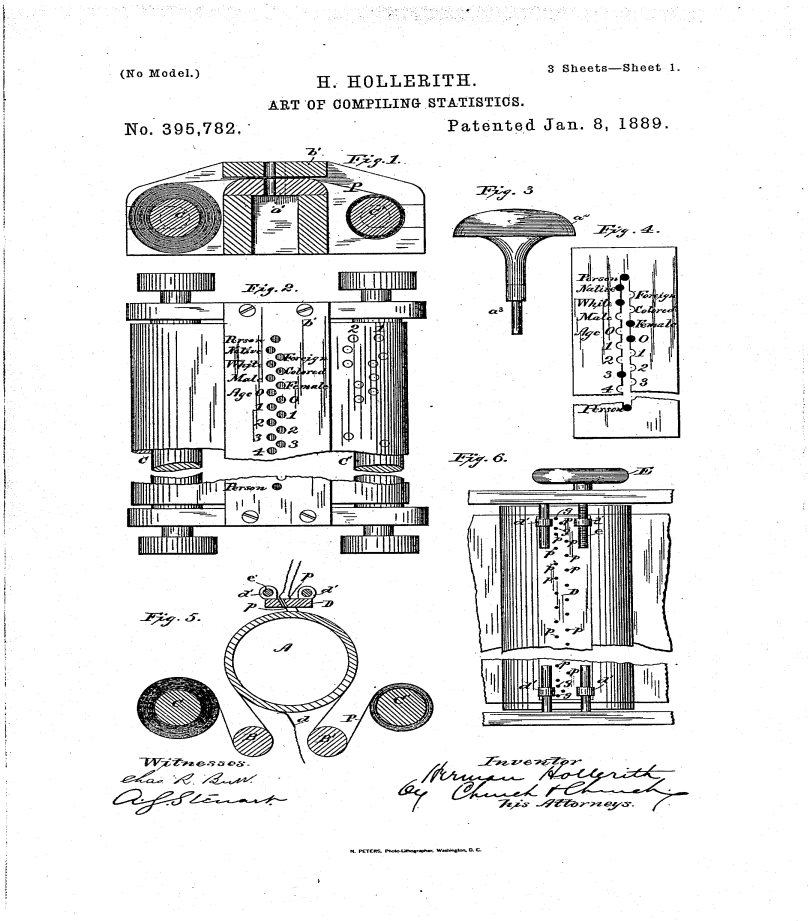
Кључне речи: базе података, везе, кључеви, дијаграми, моделирање, UML, Rational Rose, операција, кардиналност.

Садржај

1. Уводна разматрања о базама података

1.1. Историја и настанак база података

Сам настанак база података везује се за Америчког статистичара *Herman-a Hollerith-a*. Он је 1884. године пријавио патент под називом *Art of compiling statistics* који је одобрен 1889. године. Овај патент се на наш језик може превести као Систем за аутоматску обраду података, а служио је за попис становништва у Сједињеним Америчким Државама. Подаци који су се налазили на бушеним картицама су се ручно убацивали у уређај за очитавање, а обрада података се односила на пребројавање. Програмирање се сводило на избор врсте пребројавања, а радило се ручним преспајањем контаката. Дотадашња обрада података пописа трајала је 10-так година, а са овим изумом време обраде се смањило на шест недеља. *Herman Hollerith* је осмислио идеју по којој се сваки становник Сједињених Америчких Држава представља низом од 80 карактера – име, годиште, итд. попуњених празним просторима да би се за сва имена обезбедила иста дужина, тако да базе података буду „поравната". Он је продао концепт своје машине и бушене картице које су служиле за чување података у статичком бироу Сједињених Америчких Држава. Коришћењем ове машине од 1890. године попис становништва у Сједињеним Америчким Државама је постао прва аутоматизована база податак, која се у суштини састојала од хиљада кутија пунтих бушених картица. *Hollerith*-ова компанија која се бавила израдом ових машина је претеча данашњег *IBM*-а. [1]


(No Model.) 3 Sheets—Sheet 1.

H. HOLLERITH.

ART OF COMPILING STATISTICS.

No. 395,782. Patented Jan. 8, 1889.

Fig. 1. Fig. 2. Fig. 3. Fig. 4. Fig. 5. Fig. 6.

Witnesses: Chas. R. Bond, Alfred Stewart.

Inventor: Herman Hollerith, by Church & Church, his Attorneys.

Слика 1.1. Систем за аутоматску обраду података [3]

Након Другог светског рада, у компанијама и владиним институцијама почели су да се појављују први електронски рачунари. Они су се често користили за неке једноставне линеарне базе података, најчешће за рачуноводство. Ипак богатији корисници ових машина захтевали су нешто напредније и више од тадашњих машина, а све то је водило до раних база података. Занимљив податак је да су ове апликације наставиле да користе *Hollerith*-ове бушене картице, незнатно модификоване у односу на оригинални дизајн. Нефлексибилност поља исте дужине, базе података покретане 80 колонским бушеним картицама, учиниле су ране рачунаре метом напада и шала. Већина првобитних база података се односила на специфичне програме написане за специфичне базе података. За разлику од модерних система који могу бити примењени на потпуно различите базе подата, ови системи су били уско повезани за базу података да би осигурали брзину на уштрб флексиилности. [1]

Системи управљања базама података су се први пут појавили током 1960-тих година и наставили су да се развијају током следећих деценија. У овом предиоду системи засновани на датотекама су и даље имали доминантну улогу. Па ипак, први системи за управљање базом података су уведени у овој деценији и коришћени су само код великих сложених пројеката, као што је то били пројекат слетања *Apollo*-а на Месец. На овај период можемо гледати као на период у коме је демонстрирана способност ових система да управљају огромним количинама података. Такође први напори да се станадардизују предузет је са формирањем *DBT Group*-е(*Data Base Task Group*), током касних 60-их година. [1]

Током 70-их година употреба система за управљање базом података је постајала комерцијална ствар. Уведени су хијерархијски и межни системи за управљање подацима, великим делом због потреба за системом који ће моћи да управља сложеним структурама података, као што су рачуни фабрика при набавци сировина, којима је било много тешко управљати преко конвенционалних метода и ови модели се сматрају првом генерацијом система за управљање базом података. Оба система су се широко примељивала, заправо неки се користе и данас. Међутим имали су неколико велиих недостатака:

- За приступ најједноставнијим подацима били су потребни изузетно сложени програми.
- Веома ограничена независност података, тако да се програми нису могли изоловати од промена у формату података.
- Није постојала ниједна широко прихваћена теоријска подлога за било који од ових модел, за разлику од релационог модела података. [1]

Да би се превазишла сва ограничења *E.F.Cood*, као и многи други, развили су модел релационих податак података током 70-их година. Овај модел, који се сматра другом генерацијом *DBMS*-а (Database management system), доживео је широку комерцијалну употребу у пословном свету током 80-их година. Овим релационим моделом сви подаци су представљени у форми табеле. Релативно једноставан програмски језик четврте генерације, назван *SQL*, коришћен је за добијање информација. Овај модел је обезбедио једноставан приступ подацима и оним људима који нису били програмери. Модел је такође показао и своју погодност за комуникацију на релацији клијент/сервер, паралерни пренос податак и употребу *GUI*-а(Graphical User Interface). [1]

Наредна деценија, тачније 1990-те се означавају као нова рачунарска ера, најпре на нивоу рачунарске комуникације на релацији клијент/сервер, а потом и стварањем складишта за податке и употребом интернет апликација, које су добијале све више на важности у овом прериоду. Као што је управљање подацима од стране *DBMS*-а постало високо примењљиво током осамдестих година , мултимедијални подаци(укључујући и графику, звук, слике, видео запис) су постали уобичајна ствар током деведесетих година. Како би се изборили са све сложенијим, током деведесетих су уведени системи окренути ка објекту, који се сматрају трећом генерацијом. Због велике потребе за организацијом огромне количине података као струкрурисаних тако и неструкрурисаних података, овај и претходи систем су у употреби и данас.

Почетком 21-ог века јављају се нови правци за развој система за базе података, као што су:

- Могућност управљања све сложенијим типовима података. Ови типови укључују и мултидимензионалне податке, који су већ добили важност у апликацијама складиштења податак.
- Децетрализоване базе података
- Примена вештачке инталигенције ће олакшати приступ подацима и необученим корисницима
- Развој нових техника и алгоритама за анализу података – анализа складишта података
- Заштита података [2]

2. Базе података

2.1. Класична обрада података

За чување и обраду података користио се систем датотека код кога су се подаци чували у скупу неповезаних датотека. Основни недостаци овакве обраде података су:

- Редуданса података, односно вишеструко памћење истих података (нпр. подаци о радницима се налазе у датотеци за обрачун зарада и у датотеци са расподелом радних места), што доводи до проблема у ажурирању али и повећању трошкова обраде података и утрошка меморијског простора
- Зависност програма од организације података тј. логичке и физичке структуре података (промена у структури датотеке тј. убацивање новог поља у запис захтева и неминовну промену програма за обраду датотеке, иако нпр. он тај нов податак не користи)
- Ниска продуктивнос у развоју информационог система (уколико се непоходни подаци налазе у различитим датотекама, на разним медијумима, са различитим физичким организацијама задивиљење и неког једноставног информационог захтева изискује значајне програмерске напоне)
- Над структуром података представљеном великим бројем независних датотека веоме је тешко развити софтверске алате за брз и ефикасан развој апликација и за непосредну комуникацију корисника са системом) [9]

2.2. Дефинисање базе података

База података се најједноставније може дефинисати као добро структуирана колекција података која постоји релативно дуго и коју користи и одржава више корисника, односно програма(апликација). Изучавању база података може се приступити са два различита, међусобно повезана аспекта у којима се оне третирају као:

- **Системи за управљање базом података** (*Database Menagement Systems*), специфична технологија обраде података, односно софтверски систем који обезбеђује основне функције обраде велике количине података
- **Модел података**, одностно специфичне теорије помоћу којих се спецификује и пројектује нека конкретна база податак или информацијони ситем [6]

2.2.1. Систем за управљање базом података(СУБП)

СУБП (*Database Menagement System*) је софтверски систем за ефикасан унос, чување, претраживање и одржавање великих количина података, који омогућава:

- Трајно чување података – као систем датотека *DBMS* омогућава чување великих количина података независно од процеса који те податке користе. Осим тога *DBMS* обезбеђује већу флексибилност коришћењем струкутре података које подржавају ефикаснији приступ веома великим количинама података

- Креирање нових база података и дефинисање новијих шема (логичке структуре података), коришћењем специјализованог језика за дефинисање података (*data-definition language*)

- Лакши приступ и измену података коришћењем моћних упитних језика (*query language* или *data-manipulation language*)

- Складиштење података са минимумом редудансе

- Коришћење заједничких података од стране свих овлашћених корисника. Пошто се податак памти само једном, више програма истовремено захтева приступ истим подацима па *DBMS* мора да овезведи да се конфликтни захтеви и нежељене интерференције програма до којих у овој ситуацији може да дође, успрешно разиђе

- Заштиту база података и то са два аспекта – очување интегритета базе података и сигурност података. Термин интегритета базе података се користи да означи тачност података у бази. Итегритет базе података може да наруши нека грешка у програму или неки системски отказ. Сигурност базе података се одности на заштиту базе података од неовлашћеног коришћења и ажурирања података.

- Могућност опоравка базе података. Неопходно је да *DBMS* преко периодичног копирања базе на “архивске меморије” и памћења трансакција тј. операција које су између два копирања догодиле, омогући ефикасан опоравак базе података после неког отказа [презентација]

2.2.2. Модел података

Моделом података је представљена логичка структура свих података у бази као и скуп свих операција које корисник може извршити над тим подацима. Фундаменталана разлика између система ових генерација је разлика у моделу података, која се у имплементацији одговарајућих СУБП рефлектује на ефикаснос приступа подацима и обраде података, продуктивност корисника, функционалност система и подршку разноразним апликацијама. Тако се у прве две генерације сврставају мрежни и хијерархијски системи, који су осамдесетих година прошлог века готово у потпуности превазиђени релационом технологијом, као трећом генерацијом СУБП. Неки од најважнијих модела података су: [4]

- **Релациони модел** – заснован на математичком појму релације. И подаци и везе међу подацима приказују се у "правоугаоним" табелама
- **Мрежни модел** – база је приказана графиком. Чворови су типови записа, а лукови дефинишу везе међу типовима записа.
- **Хијерархијски модел** – специјални случај мрежног модела. База је приказана једним стаблом или скупом стабала. Чворови су типови записа, а хирерархијски однос изражава везу међу туповима записа.
- **Објектни модел** – инспирисан је објекто-орјентисаним програмским језицима. База је скуп објеката који се састоје од својих интерних података и "медтода" за руковање тим подацима. Сваки објекат припада некој класи. Између класа се успостављају везе наслеђивања, скупљање, односно међусобно коришћење операција. [5]

3. Пројектовање базе података

3.1. ЕР модел

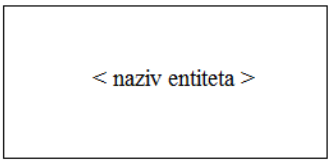
Постоји више модела за моделовање базе података. Најпознатији модел је модел P.Chena који се назива *ER-ERA* (Entity Relationship Atribute) или на нашем језику преведено МОВ(Модел Објекти Везе). Разлози због којих се највише користи су:

- Једноставан
- Прегледан
- Лак за примени у комуникацију
- Има добру семантику
- Не зависи ни од једног конкретног система за управљање базом података
- Може се релативно лако превести у било који модел базе података (хијерархијски, мрежни, релациони)[9]

Основни елементи ЕР модела су:

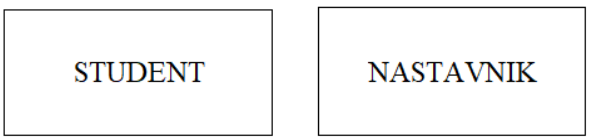
3.1.1. Ентитети

Објекти модела који су једнозначно одређени називају се ентитетима. Ентитети се графички приказују правоугаоником у оквиру кога се уписује назив типа ентитета у једнини. [9]



Слика 3.1. Пример ентитета

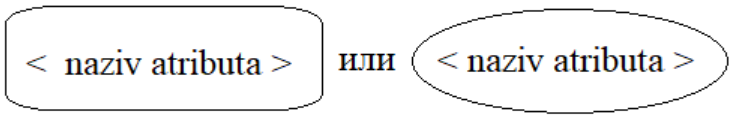
Пример: Студент и Наставник



Слика 3.2. Ентитети студент и наставник

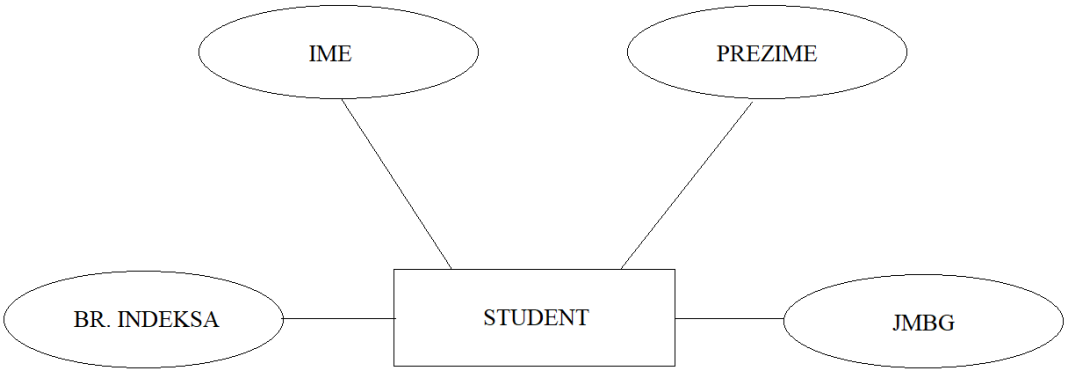
3.1.2. Атрибути

Ентитети поседују особине (својства, обележја). Опис једне особине се састоји од атрибута (пример: висина, тежина, боја...) којим је једнозначно одређене висина особине и вредност атрибута (пример: 190cm, 76 kg, зелена...). Заједно, атрибут и вредност атрибута јесу опис одређене особине ентитета. Графички приказ атрибута: [9]



Слика 3.3. Пример атрибута

Пример ентитета Студент са одговарајућим атрибутима:



Слика 3.4. Атрибути ентитета

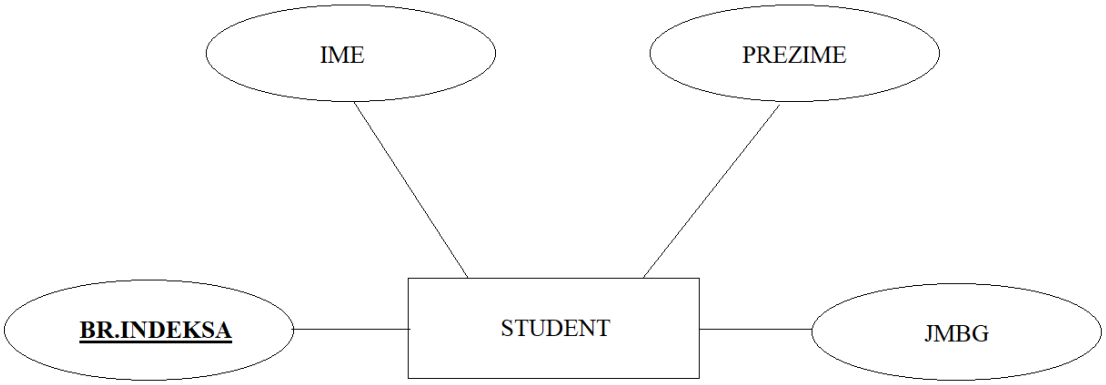
Треба разликовати појаве ентитета (*entitity instance, entity occurrence*) који представља конкретне објекте (пример: студент Новаковић, бр.инд. 70/90, ЈМБГ 1108971720025) и типове ентитета (entity class, entity set) који представља скуп свих ентитета (појава ентитета) са заједничким атрибутима (пример: скуп свих студената односно апстрактни тип ентитета STUDENT са општим атрибутима: Број интекса, Име, Презиме, ЈМБГ). Често се и један и други називају ентитетом, а из самог примера се види да је реч о конкретном или апстрактном објекту.

Што се атрибута тиче, дефинишемо још и вредност атрибута (atribut value) – појединачна појава неког атрибута (нпр. 70/90 за атрибуте Броја индекса); и домен атрибута (atribut domain) – скуп или интервал свих могућих вредности неког атрибута(пример: скуп свих бројева индекса студената уписаних на Факултет инжењерских наука у Крагујевцу или скуп свих смерова (ПМАУ, ИНФ, ПМ, ЕПТ, МКМ...) ако се као нови атрбут ентитета Студент уведе и Смер који студира).[9]

3.1.3. Кључеви

Кључ је атрибут или комбинација атрибута који једнозначно одређују ентитет из скупа. Обично је то један (прост или елементарни кључ) или комбинација два или више атрибута (сложен кључ).

У једном ентитету може постојати више комбинација атрубута који задовољавају дефиницију кључа (пример: за једно волзило могући атрубут – кандидат за кључ који једнозначно дефинишу свако возило су: регистарски број, број саобраћајне дозволе, број мотора, број шасије, итд. Сви ови атрибути појединачно одређују јединствени примерак ентитета Возило). Један од кандидата који се избре да практично служи за једнозначну идентификацију ентитета се назива примарни кључ (primary key), а сви остали, неизабрани кандидати се тада називај алтернативни кључеви.
Графички, примарни кључ се од осталих атрибута разликује по томе што је подвучен. [9]



Слика 3.5. Пример кључа

Особине примарног кључа су:

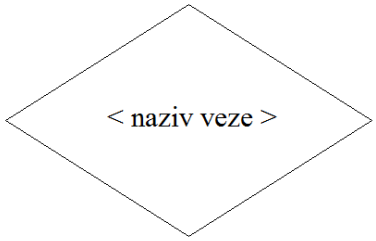
- **Једнозначност** – две појаве ентитета у склопу не смеју имати исту вредност примарног кључа без обзира да ли је кључ прост или сложен
- **Минималност –** ако је кључ сложен и ако се из њега изостави било који атрибут, губи се особина једнозначности.

За кључ се често, у недостатку јединственог идентификатора, уводи нови – ID број који се додељује свакој појави ентитета водећи рачуна да нема дуплирања.
Пример: за ентитет Предмет не усвајамо атрибут Назив као кључ, јер могу постојати два предмета са истим називом али различитим програмима и фондом часова. Зато се као нови идентификатор(кључ) уводи Šifra која се једноставно додељује сваком предмету. Слично, можемо увест Šifru ID као атрибут (и кључ) ентитета Наставник. [9]

Слика 3.6. Пример са шифром као атрибут

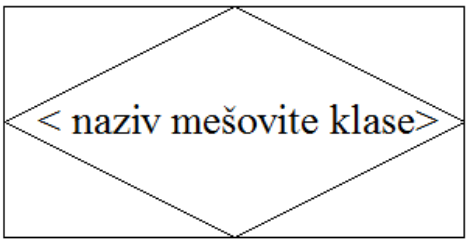
3.1.4. Везе(односи) између ентитета

Ентитети који су елементи истог скупа, као и ентитети из различитих скупова могу међусобно стајати у различитим односима тј. повезани различитим везама.
Графички се веза приказује ромбом у коме се уписује назив везе: [9]



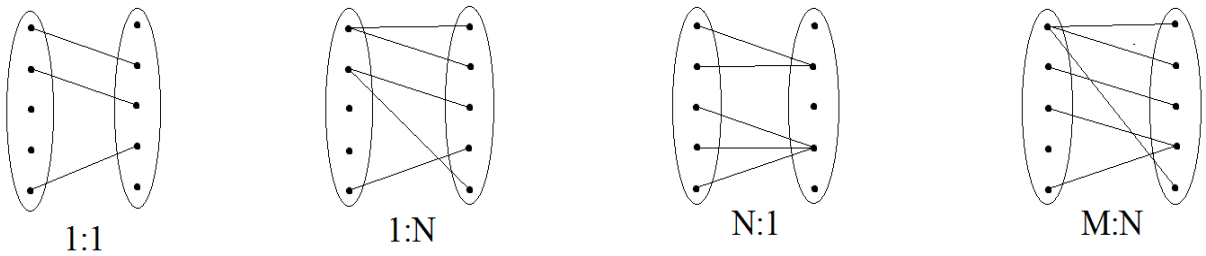
Слика 3.7. Пример везе

Представљање везе ентитетом (добија се мешовита класа ентитет – веза) уводи се да би се решио управо проблем представљања везе између веза. Врста везе се тада представља као класа ентитета што омогућава да се може успоставити веза између релација. Графички се оваква класа представља на следећи начин:[9]



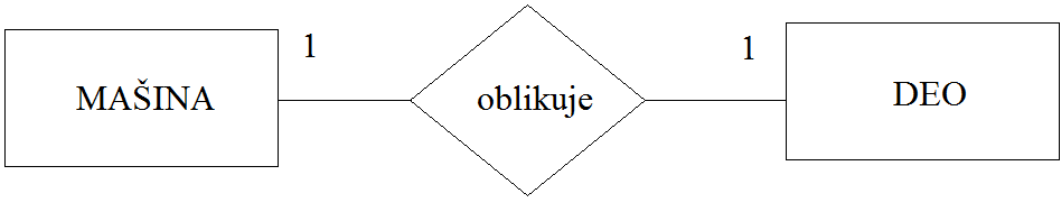
Слика 3.8. Пример мешовите класе

Степен релације, кардиналност, односно степен везе између два ентитета је веома важна особина везе. Постоје три различита степена веза које могу постојати између два ентитета:



Слика 3.9. Степени веза [9]

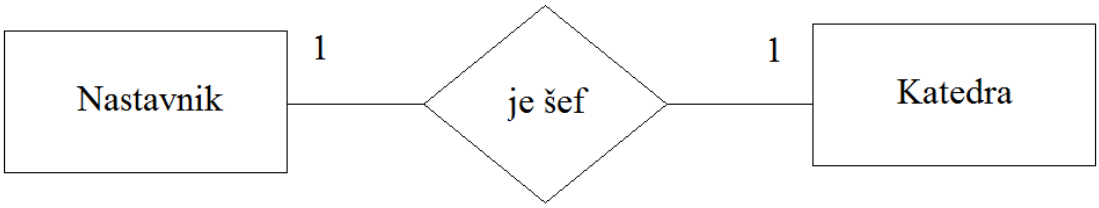
Веза 1:1(један према један), означава да се један ентитет једног типа придружује једном ентитету другог типа. [9]



Слика 3.10. Веза 1:1

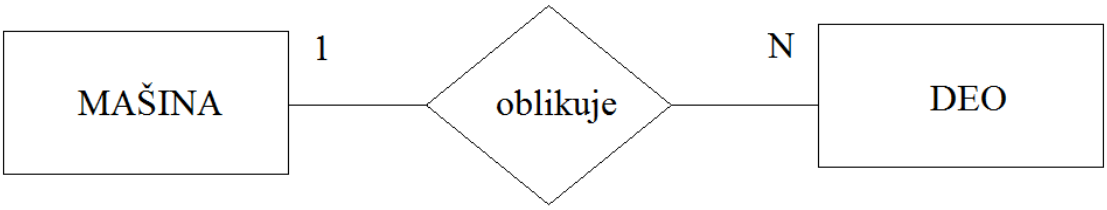
Веза 1:1 означава да једна машина обрађује само један део, а један део се обрађује само на једној машини.

Пример: између ентитета Nastavnik и Katedra се може успоставити веза Nastavnik је шеф Katedre. Јасно је да је ова веза 1:1 јер један наставник може бити шеф само једне катедре, и обрнуто, катедра може имати само једног шефа.



Слика 3.11. Још један пример везе 1:1

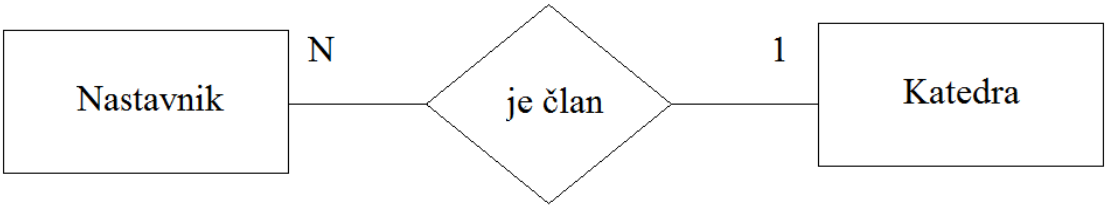
Веза 1:N(један према више), означава да се један ентитет првог типа придружује већем броју ентитета другог типа, а један ентитет другог типа само једном ентитету првог типа.[9]



Слика 3.12. Веза 1:N

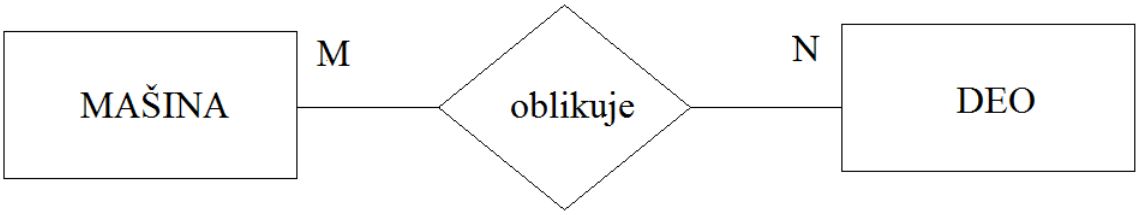
Веза 1:N означава да једна машина обрађује више делова, а један део се обрађује на једној машини.

Веза N:1 (више према један) – Посматрајмо сада и два ентитета: Nastavnik и Katedra. Ако посматрамо везу Nastavnik је члан Katedre онда је јасно да је то веза N:1 јер један наставник може бити члан само једне катедре, али једна катедра може имати више чланова. [9]



Слика 3.13. Веза N:1

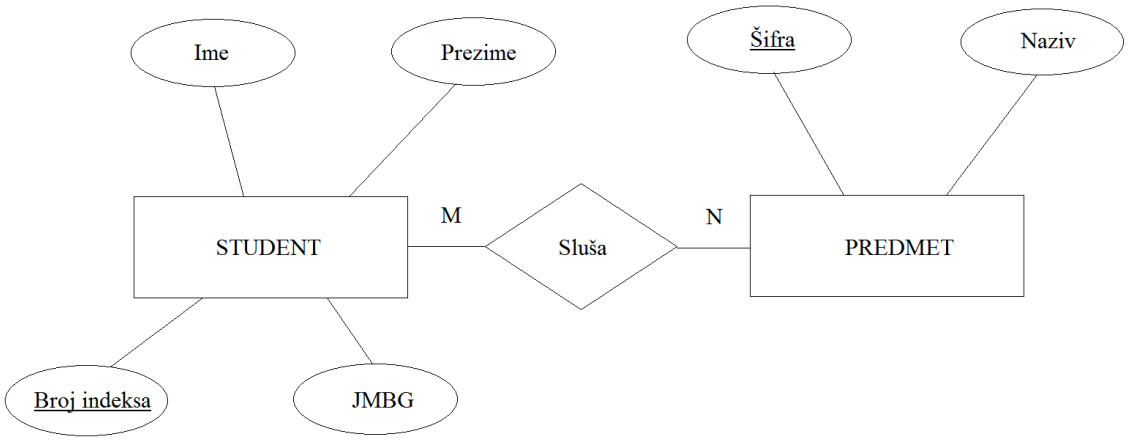
Веза M:N (више према више), означава да се једном ентитету једног типа придружује више ентитета другог типа или обрнуто.[9]



Слика 3.14. Веза M:N

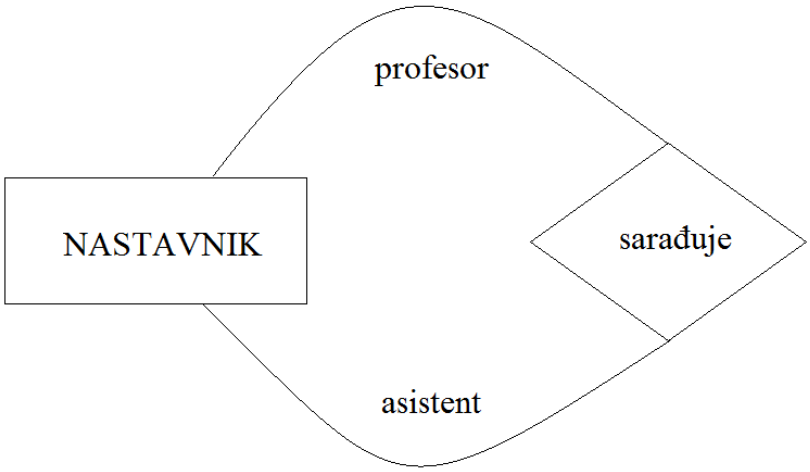
Веза M:N означава да један машина обрађује више делова, а један део се обрађује на више машина.

Пример: посматрајмо ентитете Student и Predmet. Очигледно је да је посматрана веза M:N јер један студент може да слуша више предмета, али и један предмет може да слуша више студената.[9]



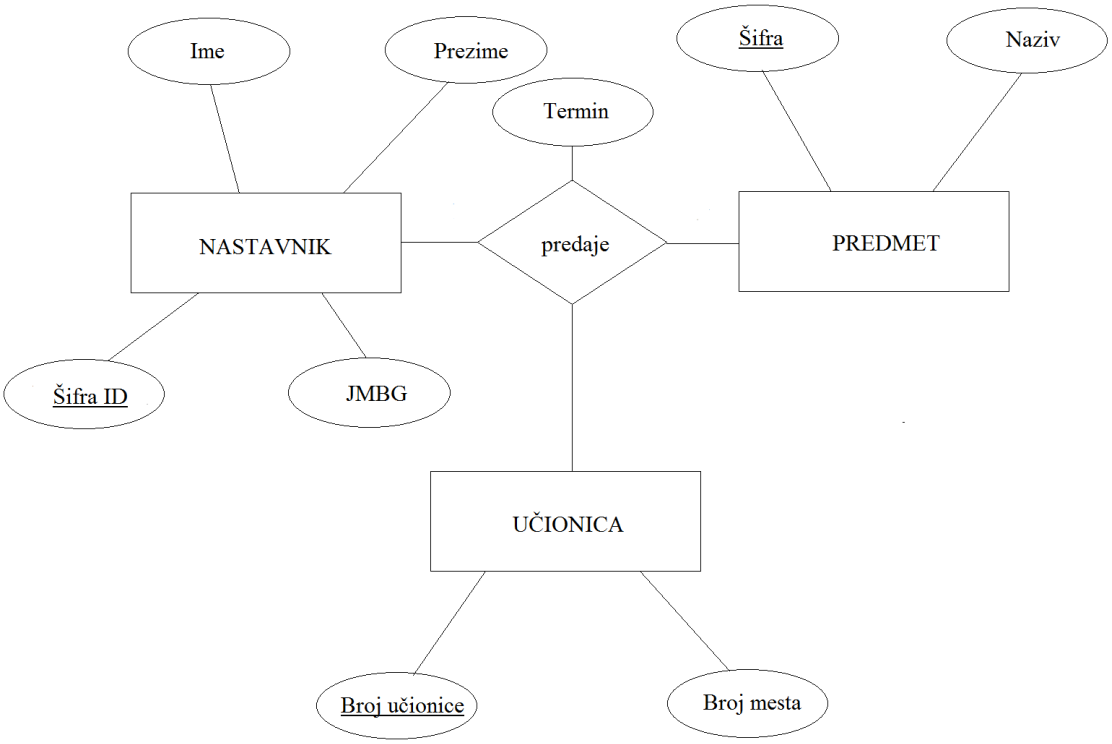
Слика 3.15. Пример ентитета са везама

Ентитети могу бити и у вези само са собом и таква веза се назива унарна. Ако нпр. ентитет Nastavnik представља у ширем смислу све запослене раднике факултета који учествују у настави(и наставнике(професори и доценти) и сараднике(асистенти и асистенит-приправници)) онда се може успоставити унарна веза Sarađuje која повезује професора са његовим асистентом, са којим сарађује у извођењу наставе.[9]



Слика 3.16. Пример унарне везе

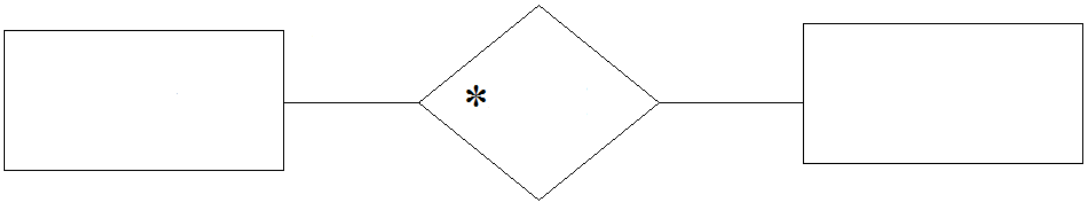
Ретко, али ипак могуће, јављају се тројне и вишеструке везе које повезују више ентитета. Пример: дефинишимо релацију Predavanje као везу између ентитета: Predmeta који се предаје, Nastavnika који држи предавање и Učionice у којој се предавање врши.[9]



Слика 3.17. Пример вишеструких веза

Релације могу имати и своје атрибуте и у претходном примеру то је Termin у коме се предавање одржава.

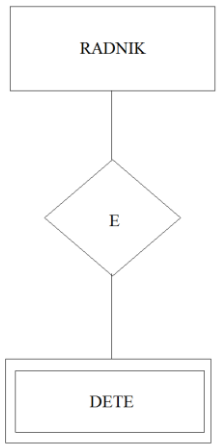
Веза може бити тотална(обавезна, мандаторна) и парцијална(опциона). Један тип ентитета има тоталну вези, ако сваки ентитет тог типа учествује у бар једној вези, иначе, веза је парцијална.(Пример: студент обавезно слуша бар неки предмет(обавезна веза), али не мора да значи да сваки предмет слуша бар неки студент(нпр. изборни предмет-опциона веза)). Тотална веза се графички означава тачком(звездицом) на темену везе према објекту који има тоталну везу.[9]



Слика 3.18. Пример тоталне везе

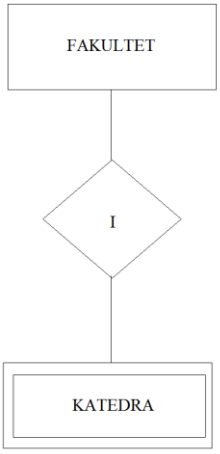
Поред до сада дефинисаних типова везе постоје и специјалне везе којима се дефинише:

- **Егзистенцијална зависност** – где је веза физичка(пример везе ентитета RADNIK I DETE. Ова веза се означава са E).[9]



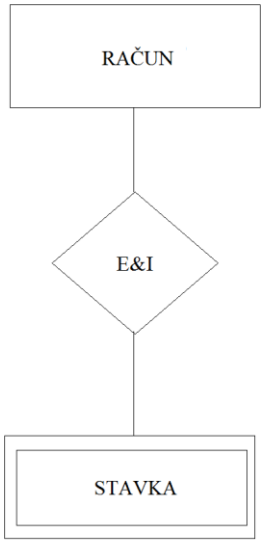
Слика 3.19. Егзистенцијална зависност

- **Идентификациона зависност** – где је кључ једног ентитета део кључа слабог ентитета. Пример везе ентитета FAKULTET и KATEDRA. Ова веза се означава са I.[9]



Слика 3.20. Идентификациона зависност

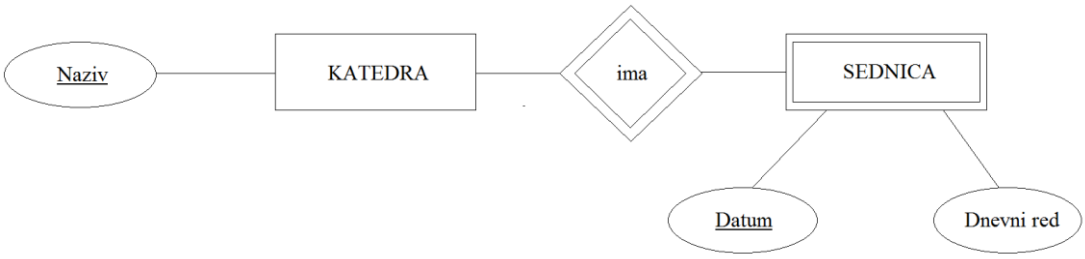
- **Идентификационо – егзистенцијална зависност.** Пример везе између ентитета RAČUN и STAVKA, означава се са E&I.[9]



Слика 3.21. Идентификационо-егзистенцијална зависност

У оквиру дефинисања специјалних веза дефинише се и појам слабог ентитета(*weak entity*) који се графички представља двоструким правоугаоником(то су у претходним примерима били DETE, KATEDRA, STAVKA). Класа слабог ентититета је врста ентитета који је на неки начин зависан од другог енитета.

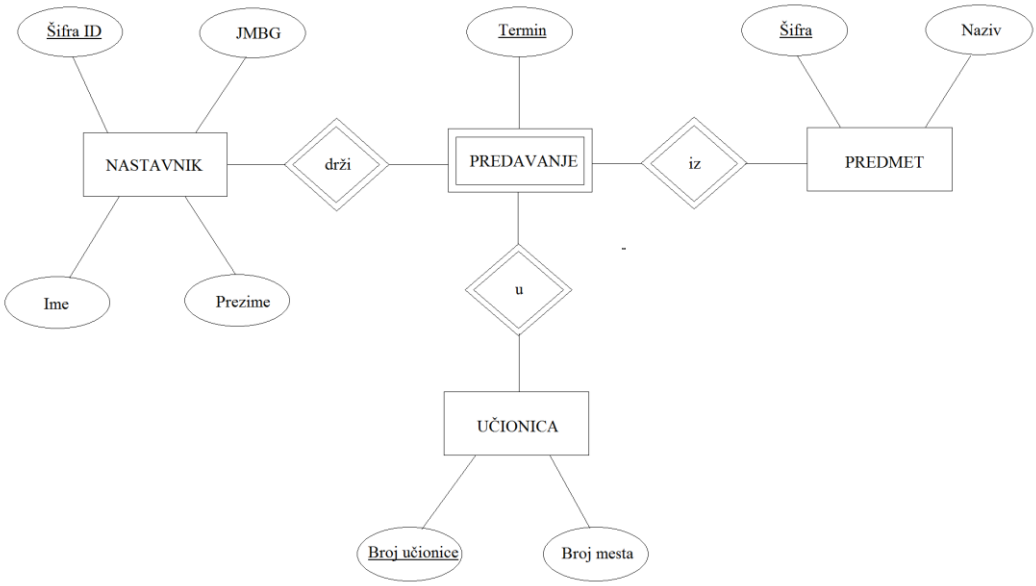
То су ентитети који за једнозначно дефинисање тј. кључ захтевају и атрибуте(примарне кључеве) других ентитета са којима су у вези (*foreign keys*). Графички, ови ентитети се представљају правоугаоницима са дуплим линијама, а њихове везе (идентификујуће везе) се у литератури означавају и ромбом са дуплим линијама. Пример: свака катедра одржава своје састанке тј.седнице које се могу описати атрибурима као што су: Datum и Dnevni red, као на слици.[9]



Слика 3.22. Слаб ентитет

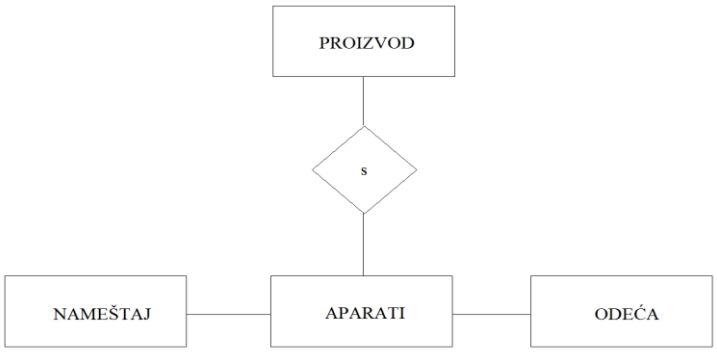
Није довољно да се за кључ усвоји Datum, јер је могуће да се две катедре одрже седнице истог дана. Такође ни комбинација (Datum, Dnevni red) није довољна јер је могуће да две катерде одрже своје састанке истог дана са истим дневним редом. Најбоље би било да се примарни кључ формира од Naziva катедре и датума састанка јер је мало вероватно да ће једна катедра два пута у току једног дана да одржи састанак. Значи, у примарни кључ ентитета Sednica улази атрибут Naziv другог ентитета Katedra, па је тако ентитет Sednica слаб ентитет.

Други извор слабих ентитета су етнитети настали заменом вишеструких веза бинарним, јер обично тако формирани ентитет захтева атрибуте других ентитета са којима је у вези за свој примарни кључ.
Пример: ентитет Predavanja је слаб јер не може бити једнозначно дефинисан само својим атирбутом Termin, већ су неопходни и атрибутни осталих ентитета.(Šifra ID да би идентификовали наставника који држи предавање, Broj učionice како би знали где се држи предавање, и Šifra predmeta који се предаје). Не морају да се усвоје сва три атрибута већ су довољна два јер ако нпр. знамо ко, када и где држи предавање знамо и из ког предмета се предавање држи(треба водити рачуна о минималности кључева).[9]



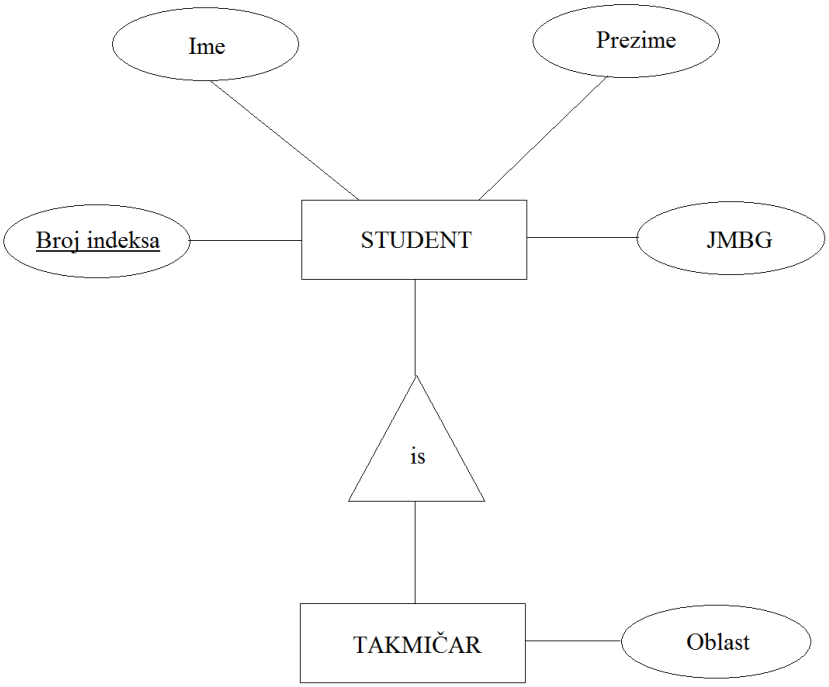
Слика 3.23. Слаби ентитети и вишеструке везе

Зависност подтипа и зависност претхођења су још два типа везе између ентитета. Генерализациона зависност или зависност подтипа је зависност између етититета подтипа и њему надређеног ентитета. Ова веза(зависност) се означава са S.[9]



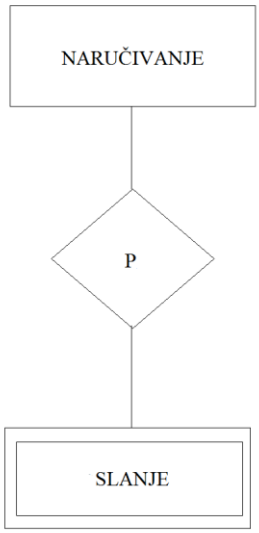
Слика 3.24. Генерализациона зависност

Ова веза се назива и “is” веза и графички се приказује троуглом. Пример: неки студенти се такмиче на Машинијади као чланови екипе факилтета за одређену облас. Дакле, сви такмичари на Машинијади су студенти, али сви студенти нису такмичари на Машинијади. Подентитет може имати своје специфичне особине, нпр. облас у којој се студент такмичи.[9]



Слика 3.25. Други начин означавања генерализационе зависности

Зависност претхођења(означава се са P) се користи за моделирање предхођења између два типа ентитета где један ентитет мора да предходи другом.[9]



Слика 3.26. Зависност претхођења

Да би се реално описао стања система, моделом података се дефинише:

- Структура податак
- Скуп ограничења
- Скуп оператора[9]

3.1.5. Ограничење модела података

Постоје дакле својства обејката, атрибура и веза коа се морају укључити у опис базе података, а нису саставни део структуре модела података.
Због тога се дефинише скуп ограничења помоћу кога се исказују дозвољена стања система и дозвољени начин преласка из стања у стање. Једна врста ограничања могу бити правила којима се специфицирају допустива стања атрибута.
(Ограничења вредности атрибута: старост радника не сме бити виша од 65 година или количина у складишту не сме бити негариван број и сл.)[9]

Оператори модела података – да би се извршила нека операција над подацима у бази података најчешће се мора прво издвојити део базе података над којим ће се извршити. Према начину издвајања разликујемо две основне врсте оператора:

- Навигационе
- Спецификационе

Навигационим операторима се бира један објекат или веза тако што се следи навигација односно логички пут кроз структуру података.
Спецификационим операторима се бира група или веза и од њих се формира нова структура на основу постојећих логичке структуре базе података.
Навигационим и спецификационим операторима се издвајају појединални подаци или мањи делови вазе податак над којим се могу извршити и сложене операције(ажурирање, издвајање вредности(мин, мах, сум), извршавање разних процедура које ће се примењивати за очување интегритета базе података).[9]

3.2 Релациони модел

Релациони модел почиње да се развија од 1970 године и то појавом *E.F.Cood* – овог чланка о релационом моделу "A Relation Model of Data for Large Shared Data Banks".

Основне карактеристике овог модела су:

- **Једноставна и природна репрезентација** (модел је представљен скупом табела, а табеле су структура најприхватљивија за комуникацију са корисником)
- **Могућа је формално – математичка интерпретација модела** (табела се може дефинисати као математичка релација, а затим искористити све предности овакве формалне дефиниције).

Ове две карактеристике односно природност описа и могућност формално – математичке интерпретације су разлог што се готово целокупна теорија модела податак и база података заснива на овом моделу.[9]

3.2.1. Структура Релационог модела

Два основна типа објекта(концепта) који карактеришу структуру Релационог модела су:

- **Домен**
- **Релација**[9]

Домен је скуп дозвољених вредности истог типа(као што је скуп Наслов књига, скуп Назива градова, скуп Датума неких догађаја, скуп Инвентарских бројева итд).

Нека су D1, D2, …, Dn (n∈N) домени (не обавезно различити). Декартов производ тих домена у ознаци D1 x D2 x …x Dn, је скуп n-торки (d1, d2, …, dn) таквих да је di∈Di за све i = 1, 2, …, n. Релација R степена n је дефинисана на доменима D1, D2, …, Dn је произвољни коначни подскуп наведеног Декартовог производа.
$R \subseteq D1 \times D2 \times \ldots \times Dn = \{(d1, d2, \ldots, dn) \mid d1 \in D1, d2 \in D2, \ldots, dn \in Dn\}$
Ако се уместо скупа индекса домена {1, 2, ..., n} узме произвољан скуп(различитих) именованих индекса {A1, A2, ... An}онда је сваки елемент di n-торке релације R дефинисан не само доменом Di већ и именованим индексом Ai. Различити именовани индекси Ai (i=1, 2, ..., n) зову се АТРИБУТИ релације R. Атрибути означавају различите улоге домена над којим је изграђена релација R. Сада се свака n-торка релације R може посматрати као скуп n парова (Ai, di), i = 1, 2, ... , n где је di вредност из домена Di.[9]

Релацијом R представља се неки тип (класа) ентитет E, док се n-торкама релације R представљају појединачни ентитети типа E, односно појављивање типа ентитета E.
Зато се сваки атрибут Ai релације R може интерпретирати као пресликавање типа ентитета E у одговарајући домен Di, tj.

$$Ai : E \rightarrow Di.$$

У терминологији релационог модела скуп Di се зове домен атрибута Ai, и означава се са дом(Ai).

Neka je e ентитет типа (класе) E. Тада је елемент di∈Di вредност атрибута Ai ентитета e (тј. одговарајуће n-торке) ако је Ai(e)=di. Слично, елементи (di1, di2, …, din)∈Di1 x Di2 x …x Din су вредности скупа атрибута {Ai1, Ai2, ... Ain} ентитета e (тј. одговарајуће n-торке) ако је (Ai1, Ai2, ... Ain)(e)=(di1, di2, …, din).

Колоне табеле одговарају атрибутима релације, а врсте n-торкама релације.

РЕЛАЦИЈА – ТАБЕЛА
АТРИБУТ – КОЛОНА
n-торка – ВРСТА

Оваква нормализована табела има следеће карактеристике:

- Нема дуплираних врста
- Редослед врста је небитан
- Редослед колона је небитан
- Све вредности у табели су атомичне[9]

Релација R се може представити у неком од следећих облика ознака односно релацијских шема:

R (A1 : D1, A2 : D2, ... , An : Dn)

(чита се: релација R чији атрибут A1 узима вредност из домена D1, атрибут A2 из домена D2 itd)

Ако су домени познати тј. када је јасно о којим доменима се ради за релацију R се употребљава ознака

R (A1, A2, ... , An),

одностно само R.

Релациона база податак је скуп скупова податак који се на логичком нивоу могу посматрати као нормализоване релације одговарајућих степена (1, 2, 3, 4, 5) и динамичког садржаја (тј. мењају се у току времена).

Код релационих база податак се разликују:

- Базне релације
- Изведене релације

Базне релације (реалне, стварне) су релације које су дефинисане независно од других релација у бази података и не могу се извести из других базних релација. На физичком нивоу представљене су одговарајућим структурама података.

Изведене релације (виртуелне, погледи) су релације које се могу извести из базних релација применом скупа операција. Немају физичку репрезентацију и представљају различите погледе које разни корисници могу имати на исту базау података.[9]

Скуп операција су: УНИЈА, ПРЕСЕК, РАЗЛИКЕ, ПРОЈЕКЦИЈЕ, РЕСТРИКЦИЈЕ, СПАЈАЊА, ДЕЉЕЊА и представљају операције релационе алгебре. Више о овим операцијама логичког пројектовања биће у оквиру упитног релационог језика *SQL* чији су саставни део.

Пример:
Тип ентитета RADNIK, према предходној нотацији означава се као релација:

RADNIK (#RADNIK, IME, PREZIME, STAZ).

Домен појединих атрибута су:
#RADNIK – скуп матичних бројева M,
IME – скуп коначних назива алфабетских знакова A,
PREZIME – скуп коначних низова алфабетских знакова A,
STAZ – скуп ненегативих целих бројева M.

Из наведеног примера се види да два стринга истог ентитета могу имати исти домен, али њихова имена морају бити различита.

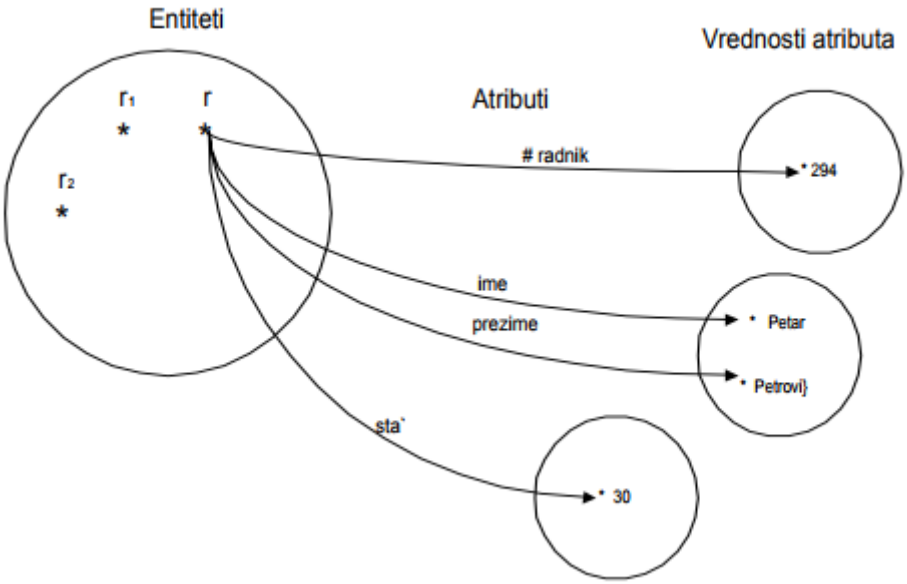
Наведена нотација RADNIK (#RADNIK, IME, PREZIME, STAZ) се интерпретира као ф-ја пресликавања ентитета RADNIK у Декартов производ домена

$$RADNIK \rightarrow M \times A \times A \times N$$

Функција која за неког раника r одређује четворку (матични број радника, име радника, презиме радника, радни стаж радника).

Два различита радника r1 и r2 имају две различите четворке и поред тога што може да се деси да постоје радници са истим именом, презименом и бројем година радног стажа, јер се матични бројеви сигурно разликују с обрзиром да се додељују тако да предстабљају јединствену идентификацију радника.[9]

На основу предходних разматрања као и да се ентитети типа (класе) E(A1, A2, ...An) могу представљати н-торкама вредности њихових атрибута, ф-ја пресликавања се помоћу Венових дијаграма може представити као на следећој слици:



Слика 3.27. Функција пресликавања [9]

а скуп ентитета класе E(A1, A2, ...An) табелом чије колоне одговарају атрибутима, а редови појединим n-торкама вредности атрибута приказано следећком табелом.
Скуп ентитета типа RADNIK предстабљен табелом:

# RADNIK	IME	PREZIME	STAZ
341	PETAR	PETROVIC	30
385	PRODAN	PRODANOVIC	10
448	DRAGAN	POPOVIC	3
1978	MARKO	MARKOVIC	15

Слика 3.28. Табеларни приказ ентитета [9]

Још неке од особина које има Релациони модел базе података:

- **Недостајућа вредност**

У бази података може постојати потреба за регистровањем недостајућих вредности. У реалности се најчешће појављују као:
- **тренутно непознате вредности** (пр. статус издавача...)
- **неприменљиво својство** (вредност атрибута ИМЕ_СУПРУЖНИКА за особу која није у браку).[9]

- **Примарни кључ**

Нека је $X \subseteq \{A1, A2, ... An\}$ подскуп скупа атрибута релације R (A1, A2, ... An). Каже се да је X кључ релације R ако су испуњена следећа два услова:

а) X функционално одређује све атрибуте релације R, тј. $X \rightarrow Ai$, za i=1, 2, ..., n.

б) ниједан прави подскуп од X нема ту особину, односно не постоји $X' \subset X$ тако да $X' \rightarrow Aj$ а ако постоји онда $X' /\rightarrow Aj$.

Свака релација RM мора имати примарни кључ.
За примарни кључ важи:
- две разне n-торке једне релације на могу имати идентичне вредности примарног кључа
- ниједан од атрибута примарног кључа релације не сме имати недостајуће вредности ни у једној n-торци.

- **Страни (секундатни) кључ**

То је атрибут(или група атрибута) која у датој релацији није примарни кључ, али је примарни кључ у некој другој релацији базе података. Спољни кључ служи за повезивање са вредношћу примарног кључа те друге релације.

- **Интегритет података Релационог модела**

Кад говоримао о интегритету РМ подразумевамо услове које треба да задовоље подаци у бази да би стање базе било ваљано. При сваком ажурирању БП, СУБП проверава задате услове интегритета. Сваки услов инегритета је неко тврђење, да би стање базе било ваљано, то тврђење мора да има вредност Тачно.
Услови интагритета могу бити
- **специфични**
- **опсти**

Специфични се односе на задати атрибут, домен или релацију (задају се експлицитно), а општи се могу применити на сваку релацију (задају се имплицитно). Пример за специфични услов интегритета (нека шифра има облик прва два знака су слова а следи троцифрени број) (Регистарска таблица) (Тираж издавача ”Просвета” не може бити мањи од 5 000)
Општи услови интегритета су:
- **интегритет ентитета**
- **интегритет обраћања** (референцијални интегритет).

Интегритет ентитета се односи на атрибут кључ који мора да има дефинисану и одређену вредност тј. његова вредност не сме бити недостајућа.
Референцијални интегритет се односи на спољни кључ релације и мора узимати вредности примерног кључа друге релације или нулл вредност.[9]

4. Визуелно моделирање

Визуелно моделирање је начин размишљања о проблемима при коме се користе модели засновани на идејама из реалног света. Модели су корисни за разумевање проблема, за комуникацију међу људима који су укључени у пројекат (корисиници, стручњаци за домен, аналитичари, дизајнери), за моделирање предузећа, за припрему докумантације и за дизајн базе програма и база података. Моделовање омогућава боље разумевање захтева, чистији дизајн и системе погоднije за одржавање.

Модели су апстракције које одсликавају кључне елементе комплексног проблема или структуре тиме што елеминишу мање битне детаље и на тај начин, чине проблем далеко лакшим за разумевање. Способност апстракције је основна људска способнос која нам помаже да се изборимо са сложеношћу. Инжењери, уметници и занатлије већ хиљадамо година користе моделе да би испробали своје концепте пре него што их спроведу у дело. Развој софтверских система у овом не би требало да буде изузетак. Да би направио комплексан систем програме мора да апстрахује различите видове система, направи моделе користећи прецизну нотацију, потврди да модели задовољавају захтеве система и постепено додаје детаље да би моделе трансформисао у реалну примену.

Моделе сложених система правимо због тога што такве системе не можемо у целини замислити. Људска способност за поимање сложености ограничена. Овај концепт се може уочити у архитектури. Уколико желите да сазидате шупу у дворишту, можете одмах почети да радите; ако хоћете да сазидате нову кућу, највероватније ће вам прво требати нацрт. Иста правила важе и у свету софтвера. Посматрањем линија кода или анализом образаца у *Visual Basic* – у програмер неће стећи општи увид у пројекат. Конструкција модела даје прилику програмеру да се усредсреди на ширу слику која му пружа увид у интеракцију појединих компоненти лишен фрагментарног представљања сваке од компонената.

Све већа сложеност, која настаје у пословном окружењу које се стално мења и у коме влада јака конкуренијца, пружа јединствене изазове програмерима система. Модели нам помажу да организујемо, визуализујемо, разумемо и правимо сложене системе. Они се користе, а и користиће се, да би нам помогли да се суочимо са изазовима развоја софтвера. [7]

5. UML (Unified Modeling Language)

UML је скуп графичких нотација који нам помажу у опису и дизајнирању софтверског система. Први и најважнији циљ *UML*-а је да је то општи језик за моделирање који сви могу користити. Не налази се ни у чијем власништву и заснован је принцима који су постигнути договором великог дела рачунарске заједнице. Планиран је тако да укључи елементе свих важнијих методологија како би га оне могле користити као свој језик за моделовање.
Дизајниран је тако да буде природан за употребу и са најлогичнијом и најјаснијом нотацијом. Прављен је да подржава једноставно изражавање добрих концепата као што су енкапсулација, раздвајање одговорности и приказивање намере везане за неку софтверску конструкцију. Обиман софтвер, дистрибуиран софтвер, конкурентан софтвер, тимски рад, на сваку од ових ставки се мислило при дизајнирању *UML*-а. [10]

5.1. UML концепти

UML концепти се могу груписати у неколико група:

- **Статичке структуре –** свака методологија мора дефинисати одређене кључне елементе структуре везане за софтвер који се прави. Тај поглед на скелет система је статички поглед на апликацију. Основни концепти софтверског система се моделују класама које нису ништа више него скупови објеката са заједничким понашањем и/или подацима. Подаци се моделирају атрибутима, док се понашање класа описује операцијама које објекти могу да изврше. Неколико класа може делити унутрашњу структуру кроз концепт генераизације при чему свака класа ниже у хијерархији може инкрементално додавати нове информације и/или операције, док наслеђује неке постојеће из виших класа у хијерархији. Објекти класа могу бити повезани на неки начин у току самог извршавања (на пример школа не постоји без ученика). Овакве повезаности се описују асоцијацијама. Осим ових основних појмова постоје и интерфејси, сигнали, случајеви употребе, типови података и сл., али ће они бити представљени касније у тексту. Статичка структура се представља дијаграмом класа.

- **Динамичко понашање –** постоје две дименизије код моделирања понашања објекта. Једна се односи на његову историју, а друга на правилности у комуникацији са другим објектима (на пример слање “откуцаја код” мрежних апликација). Историја објекта без комуникације са другим објектима се може једноставно представити стањима кроз које објекат пролази. Он прави неке акције и мења стања у складу са својим акцијама и спољашњим одговорима на те акције. Друга димензија приказује како неки објекти заједно сарађују да би остварили неки већи посао. Ту треба да се види који објекти су колико присутни (од почетка посла до краја или само на почетку итд.) и какве поруке они међусобно размењују.

- **Имплементациони конструкти** – такође, осим логичке подршке *UML* пружа и подршку самој имплементацији решења. Дефинише компоненту која мора имати одређени јавни интерфејс. Основна идеја компоненте је да буде лако заменљива другом која пружа исти интефејс (наравно неком модернијом и бољом имплементацијом тог интерфејса). Често у апликацијама имамо удаљене ресурсе који се користе (на пример сервер са базом података није исти као и апликативни сервер). Ови ресурси се називају чворови. Постоји тзв. продукциони поглед на апликацију (Енг. деплоумент виењ) који приказује организацију ових компоненти по чворовима, топологију чворова, модуће миграције компоненти и сл.

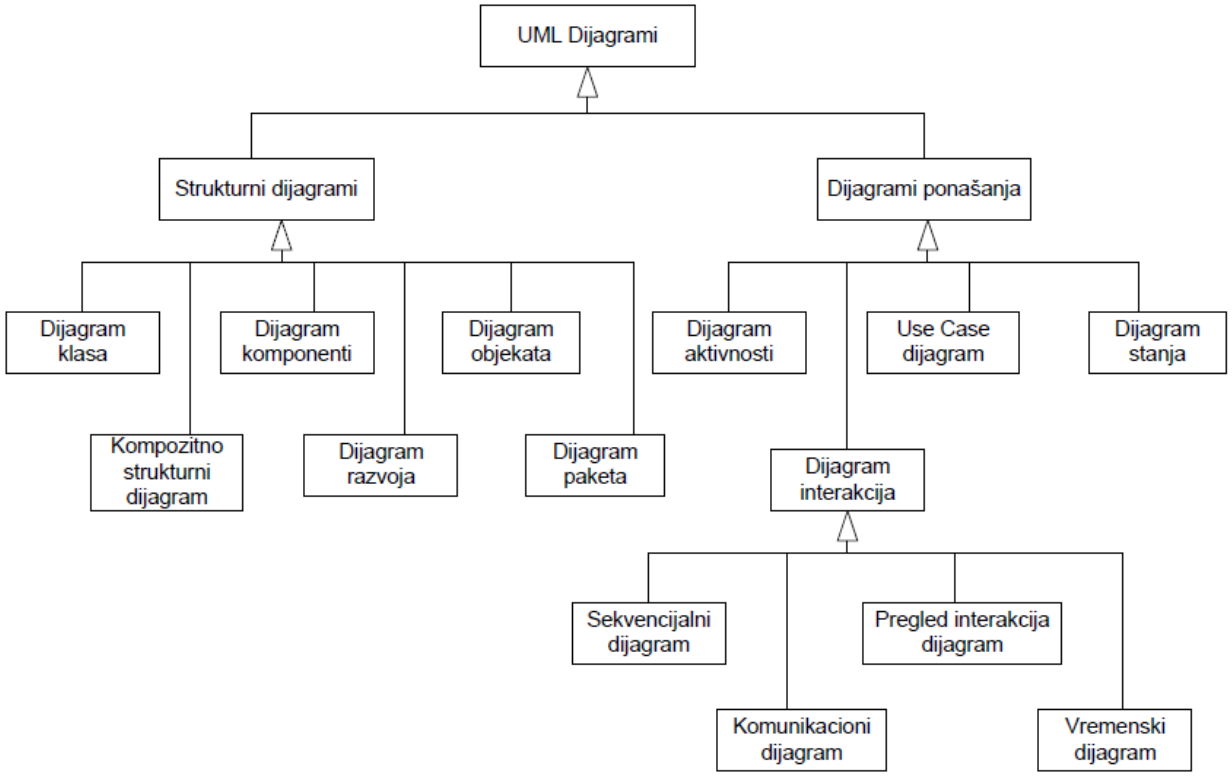
- **Организација модела** – мало нелогичан назив који описује једну јако природну потребу – тимови морају да могу да раде на различитим деловима система конкурентно. *УМЛ* пружа подршку подели посла на пакете, управљању њиховим међузависностима и сл.

- **Механизам проширења –** аутори *UML*-а су били свесни динамичког развоја рачунарства који се дешаваи тада невиђеном брзином и знали су да ће једног дана доћи до потребе за проширењем *UML*-а. Због тога су обезбедили одређене механизме којима се то може извести без опасности од доласка до неких радикалнијих промена.[10]

5.2. Врсте UML дијаграма

У процесу развоја софтвера појављују се разне улоге које користе различите технике формирања дијаграма, за решавање различитих проблема. То у ствари значи да бројни учесници у процесу развоја „разговарају“ истим језиком. Стандардизацијом и општим прихватањем *UML*-а створени су предуслови да све активности које се не могу реализовати помоћу њега биду усаглашене са активносима који се реализују помоћу њега. Основна подела *UML* дијаграма је на следеће три групе[10]:

- Структурни дијаграми
- Дијаграми понашања
- Дијаграми интеракције

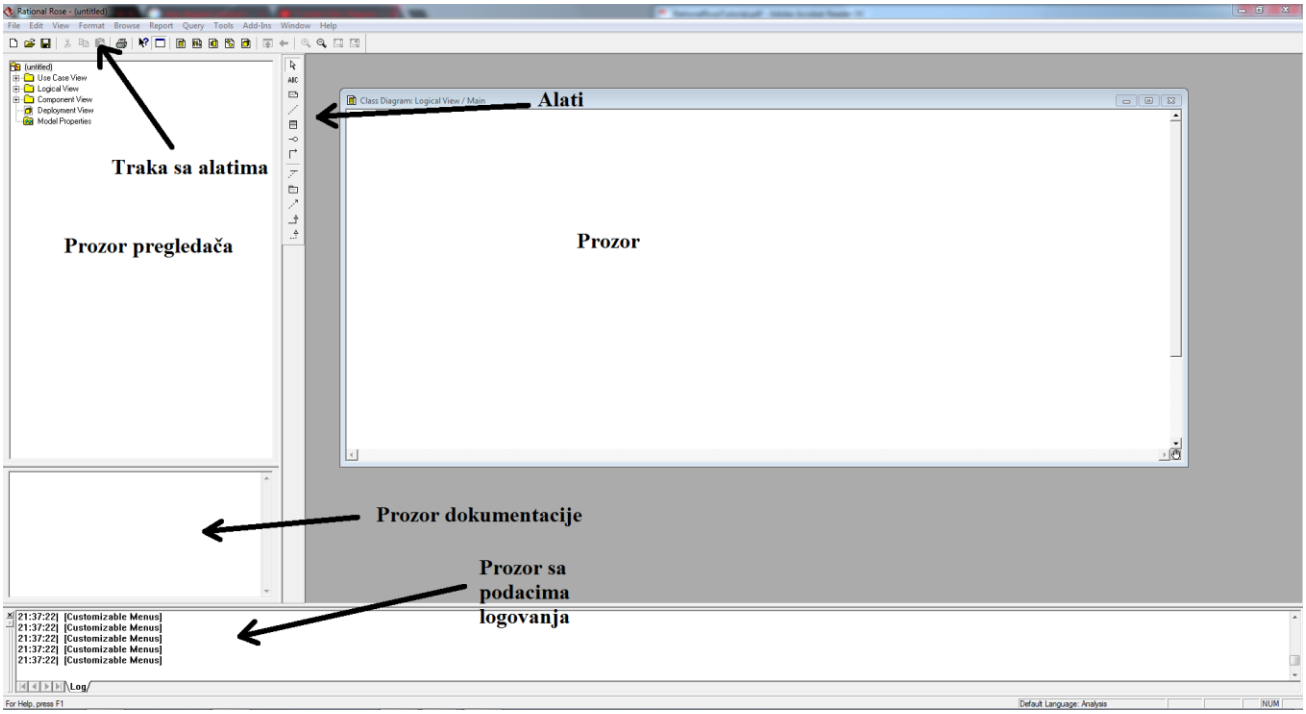


Слика 5.1. Врсте UML дијаграма [10]

6. Програмски пакет Rational Rose

Најбоље је да сваки метод развоја софтвера буде подржан неком алатком. У самим почетцима алатке су биле папир и оловка док данас на тржишту постоје многе алатке почев од једноставних алатки за цртање до софистицираних алата за објектно моделирање. Фамилија програма *Rational Rose* пројектована је тако да програмерима обезбеди комплетан алат за визуално моделирање који омогућавају развој робусних, ефикасних решења за стварне пословне потребе у сложеним системаским окружењима као што су клијент/сервер, дистрибуирана предузећа и рад у реалном времену. Производи *Rational Rose* користе општи, универзални стандард и тиме приближавају моделирање како програмерима који моделирају логику апликација, тако непрограмерима који би хтели да моделирају пословне процесе.[7]

Rational Rose је објектно – орјентисани UML (Unified Modeling Language) алат за дизајн софтвера намењен визуелном моделирању и компонентној конструкцији софтверсих апликација на нивоу предузећа. Примери модела укључују креирање актера, класа, веза, објеката, веза, итд. *Rational Rose* користи класичне UML концепте за графичко моделирање софтверских апликација. Ово олашава документовање окружења, захтева и укупног дизајна. Ово је моћан GUI(Graphical User Interface) алат за ефикасан и повољан кориснички систем који користи такозвани “drag and drop” начин за маневрисање. Одређене верзије овог алата производе релевантан изворни код за дизајнерске моделе.[8]
За почетак ћемо објаснити поједине делове програмског пакета Rational Rose, а касније и сам поступак креирања једног од дијаграма у овом програмском алату.



Слика 6.1. Графички интефејс програма

Трака са алатима:
Алати који се овде налазе су увек приказани независно од врсте покренутог дијаграма и када смо у програму довољно је да поставимо курсор на неку од иконица и да добијемо њен назив.

Прозор прегледача:
Хијерархијски навиганиони алат који могоућава да видимо имена и иконе које представљају дијаграме и елементе модела. Ознака + показује да се иконица може проширити да би смо добили више информација. Ако стоји – значи да је стабло максимално проширено. У случају да овог прегледача нема могуће га је покренути из *View* менија.

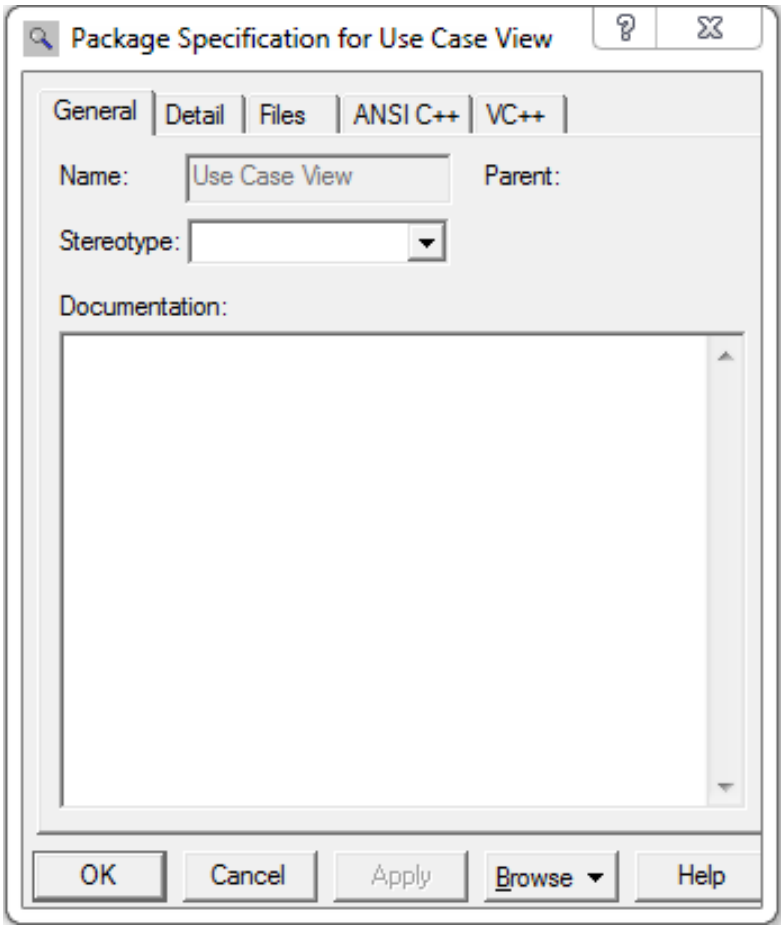
Прозор:
Омогућава нам да креирамо, апдејтујемо различите моделе дијаграма који су графички представљени.

Трака алата дијаграма:
Јединствена је за сваки дијаграм и може бити модификована. Приказује се само када је активан дијаграм који користимо. Може бити видљив или сакривен и као и стандардна трака са алатима ако се стави курсор преко иконе може се видети њен назив.
Пример алата који се користе за креирање дијагрма класа[11]:

Alati		Namena
	Selection Tool	Vraća kursor na strelicu i omogućava selekciju podataka
ABC	*Text Box*	Dodaje polje za tekst na dijagram
	Note	Dodaje napomenu na dijagram
	Anchor Note or Item	Povezuje napomenu sa slučajem upotrebe ili sa akterom na dijagram
	Class	Dodaje klasu na dijagram
	Interface	Dodavanje novog interfejsa klase
	Unidirectional Association	Crtanje unidirekcione asocijacije
	Association	Crtanje asocijacije
	Aggregation	Crtanje agregacije
	Association Class	Veza klase asocijacije sa odnosom asocijacije
	Package	Dodaje paket na dijagram
	Dependency or Instantiates	Crtanje odnosa zavisnosti
	Generalization	Crtanje odnosa generalizacije
	Realize	Crtanje odnosa realizacije

Слика 6.2. Трака са алтаима

Прозор спецификација:



Слика 6.3. Прозор спецификација

Овај прзор се отвара тако што се десним тастером миша кликне на Use Case View па се иде на Open Specification. Представља текстуалну презентацију елемената модела која омогућава манипулацију својствима елемената. Мора се имати у виду да се информације додате у прзор документације аутомацки додају у поље за документацију у прозору спецификације.

Прозор са подацима логовања:
Приказује напредак, резултате и грешке, а ако притиснемо десни тастер миша можемо видети расположиве акције.[11]

6.1. Погледи

Као што постоје многи погледи на кућу у изградњи, мрежног дијаграма, плана надморске висине, тако постоје и многи погледи на развој софтвера.
Rational Rose је организован око следећих погледа софтверских пројеката:

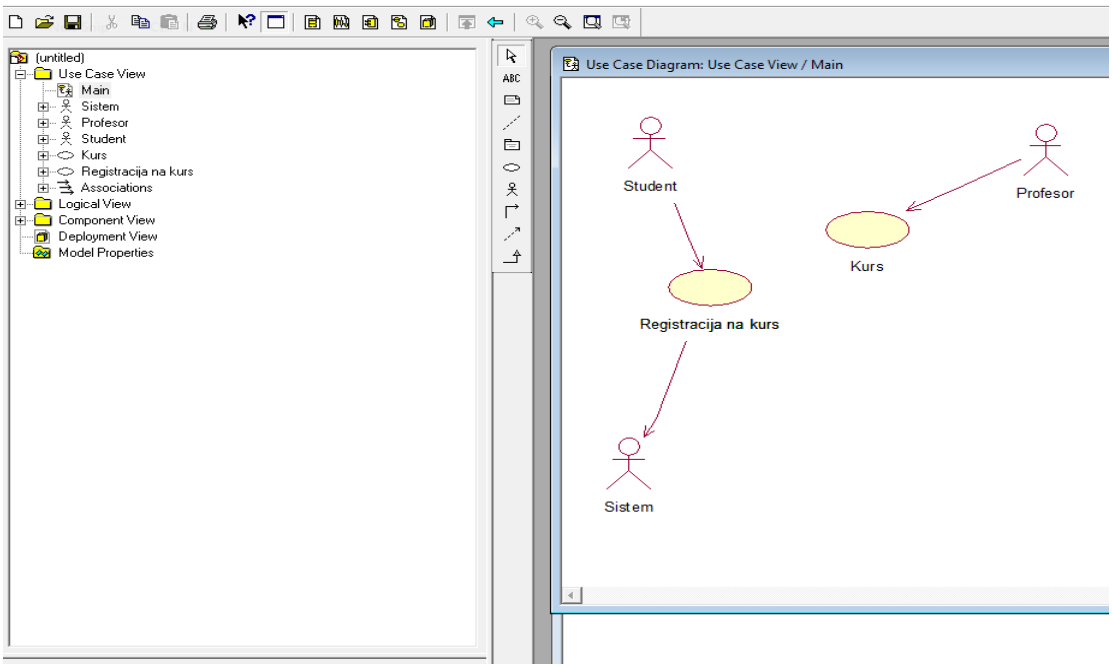
- **Use Cese**
- **Logical**
- **Component**
- **Deployment**

Сваки од ових погледа презентује различити аспект модела који ће у наставку бити објашњени.[11]

Поглед случајева употребе (Use Case view)
Овај поглед помаже у разумевању и коришћењу система(које су функције система), а дијаграмо који се користе су:

- Use Case diagrams
- Sequencal diagrams
- Collaboration diagrams
- Activity diagrams

Овај поглед садржи главни дијаграм,а остали дијаграми се могу добити преко анализе и дизајнерског процеса.[11]

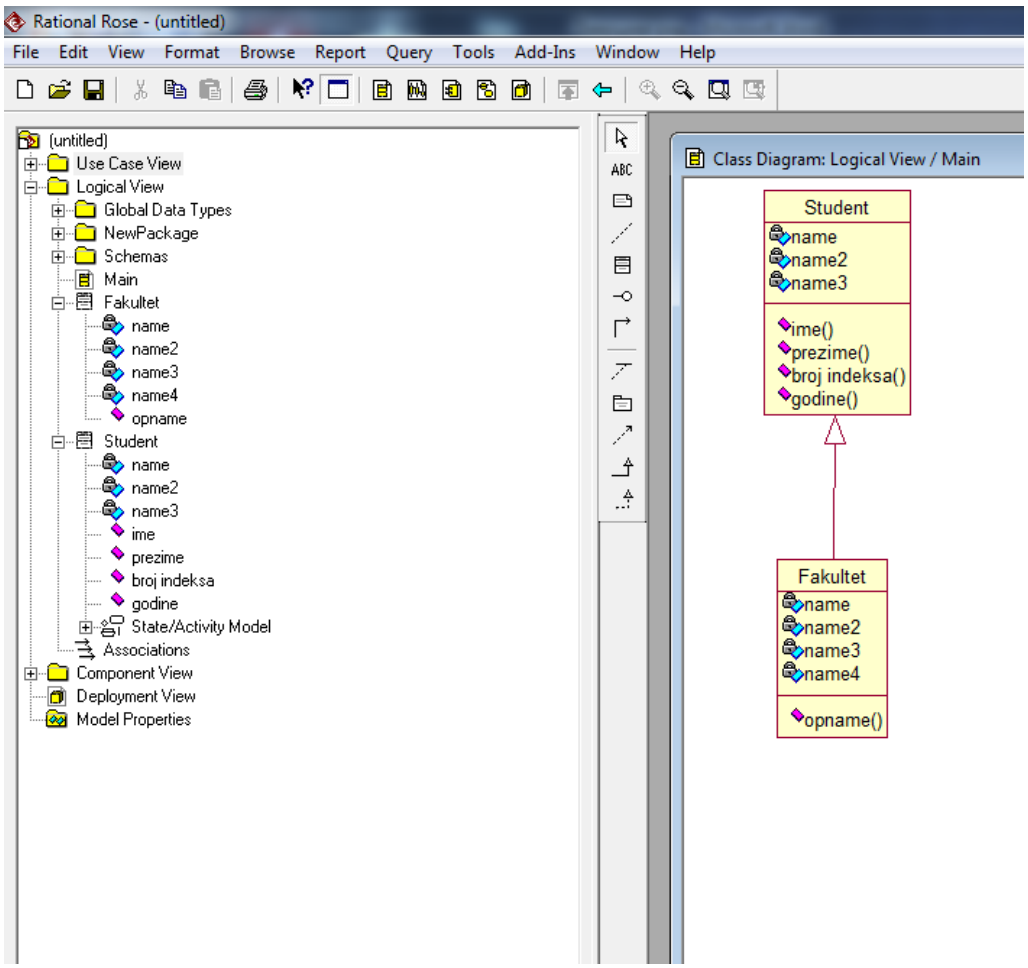


Слика 6.4. Поглед случајева употребе

Логички приказ (Logical view)
Логички приказ се односи на функционалност захтева система. Овај поглед разматра класе и њихове везе. Дијаграми у овом погледу су:

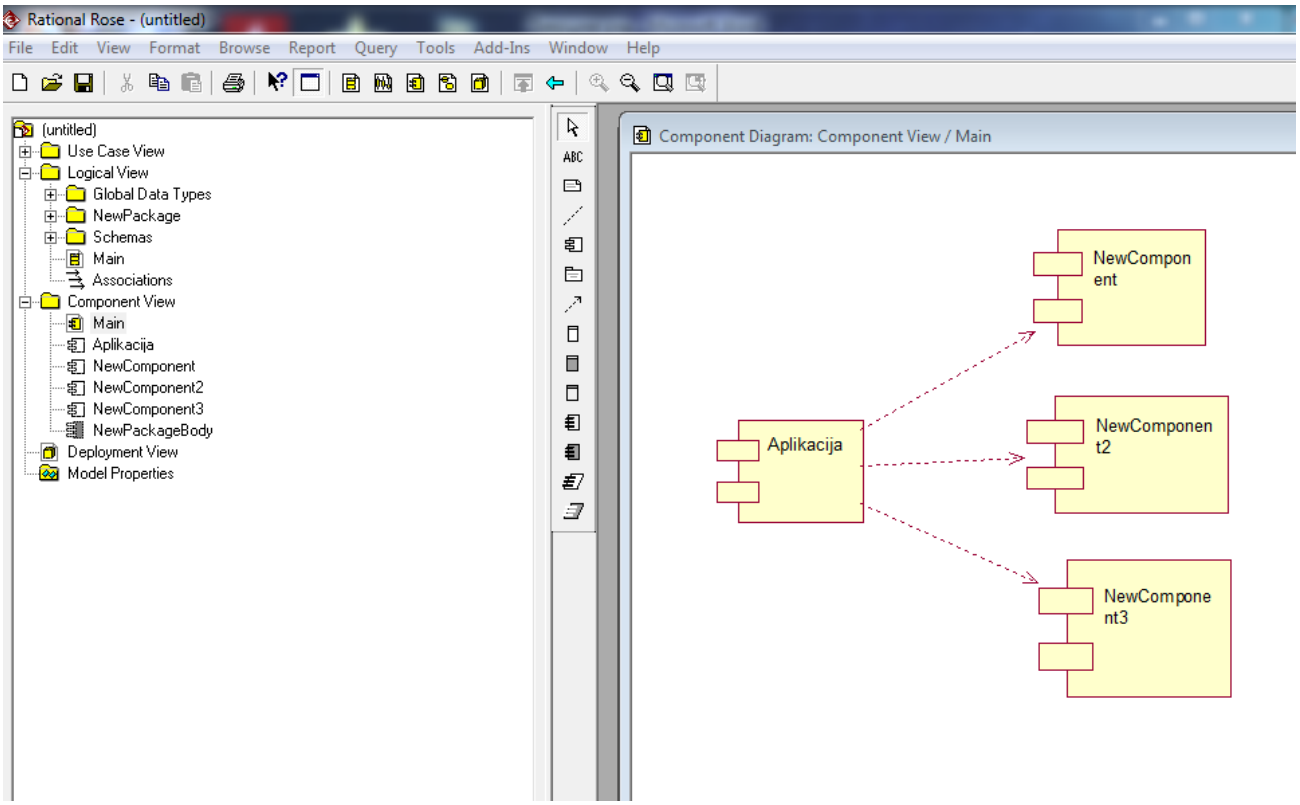
- Class diagrams
- Statechart diagrams

Овај поглед садржи главни дијаграм,а остали дијаграми се могу добити преко анализе и дизајнерског процеса.[11]



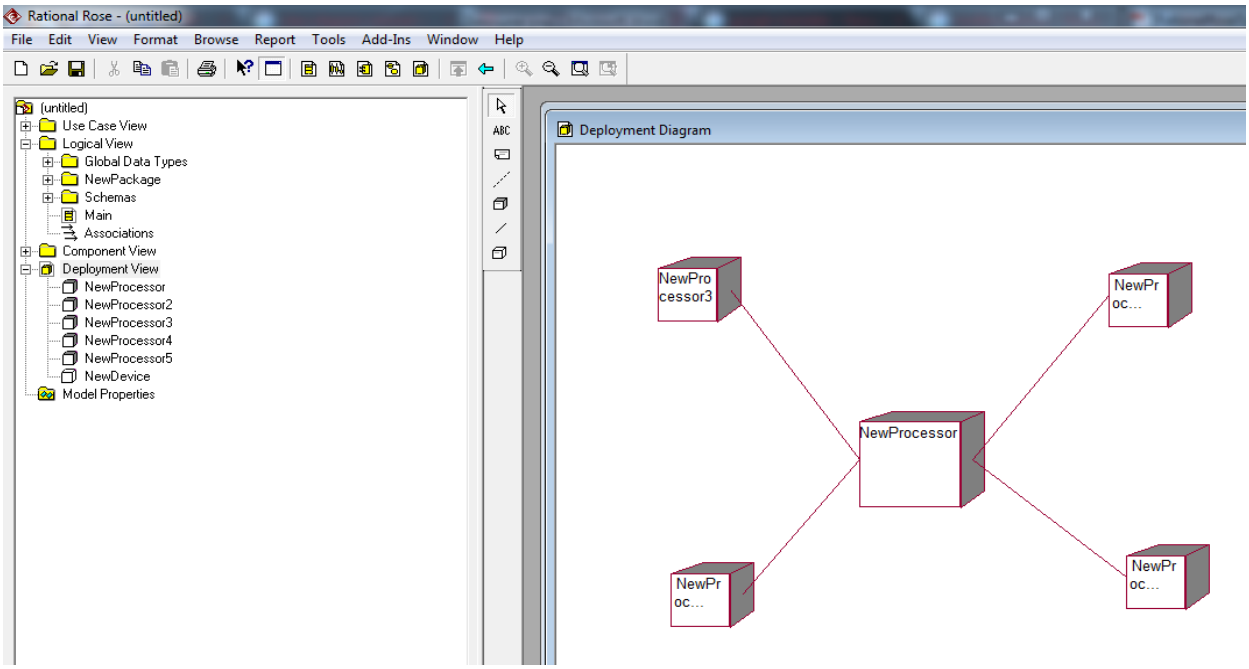
Слика 6.5. Логички поглед

Приказ компоненти (Component view)
Пример компоненти се односи на софтверску организацију система. Овај поглед садржи информације о софтверу, извршењима и библиотекама компоненти система, такође поседује само дијаграме компонената. Садрже компоненте и зависности и користе се да би се конструисао софтвер система. Компоненте представљају физичко паковање модула кода (извора кода фајла, библиотеке, динамичке компоненте или програме који моду га се изврше).
Овај поглед садржи главни дијаграм,а остали дијаграми се могу добити преко анализе и дизајнерског процеса.[11]



Слика 6.6. Приказ компоненти

Приказ распорда (Deploymetn view)
Приказује мапирање процеса на хардвер. Овај тим дијаграма је најкорисније у дистрибутивним архитектонским окружењима где може да се деси да су нам апликације и сервери на различитим локацијама. Показује како изгледа систем и где можемо инсталирати софтвере.[11]

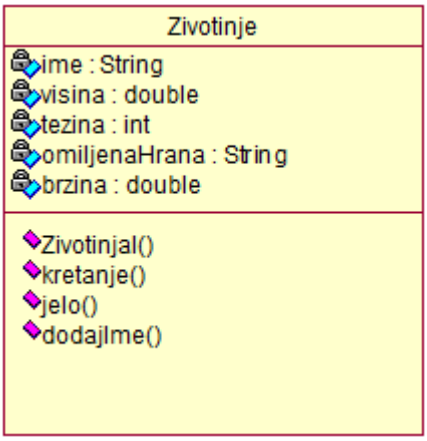


Слика 6.7. Приказ распореда

6.2 Дијаграм класа (Class Diagram)

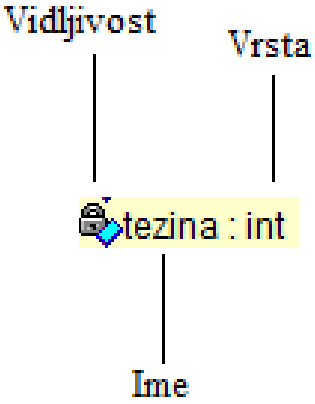
Као што је раније могло да види постоје разне врсте UML дијаграма који су подељени у неколико група. У даљем делу овог рада биће објашњено креирање дијаграма класа(Class Diagram) у Rational Rose – у који представља један од дијагрма који се могу направити у овом програму.

Класа је опис групе објеката са заједничким својствима(атрибутима), понашањем(операцијама), релацијама са другим објектима и семантиком. Према томе, класа је шаблон за стварање објеката. Сваки објекат је конкретан примерак неке класе и не може бити примерак више од једне класе. А дијаграми класа објашњавају ове класе односно како су оне повезане.[11]



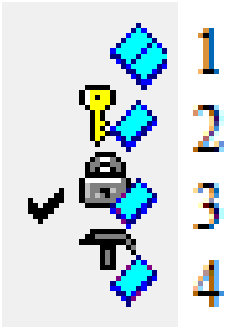
Слика 6.8. Пример класе

Узета је за пример класа животиње, овде се могу видети преходно поменута својства(атрибути) и понашања(операције). Својства тј. атрибути представљају податке и вредности који ближе одрђују једну класу, док својства одностно понашања представљају оно што објекат може да уради или оно што се може урадити том објекту. На слици испод може се видети шта који део представља.



Слика 6.9. Приказ шта који део претставља у класи

Као што се може да видети сва својства и понашања имају одрђену видљивост која показује коме су подаци доступни што се може боље видети на слици испод.



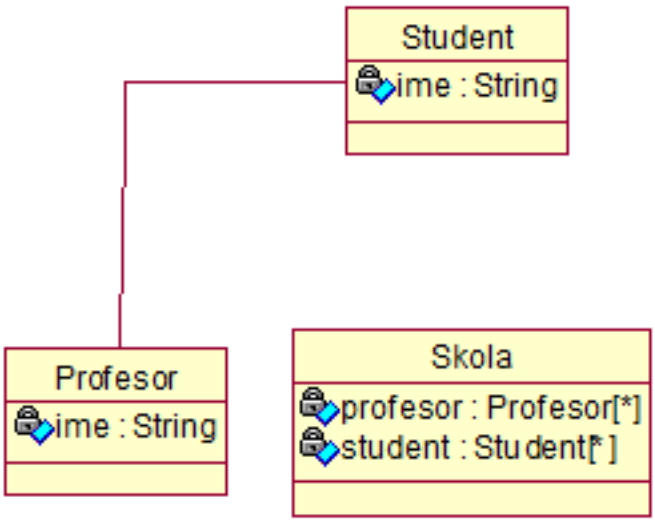
Слика 6.10. Видљивост

- Иконица испред броја 1 показује да је видљивост доступна свима (*Public*).
- Поље број 2 показује да је предмет заштићен (*Protected*) и да му се може приступити методом из исте класе или подкласе које је креирана из те класе.
- Поље број 3 показује да је предмет приватан(Private) односно да је доступан само осталим предметима који се налазе у истој класи.
- Поље број 4 показује да предмет може бити доступан свим класама које се налазе у истом пакету али ова видљивост се ретко користи (Package/Default) [11]

Везе које се користе при креирању дијагрма су:

- *Association* (Асоцијација)
- Aggregation (Агрегација)
- Unidirection Aggregation (Унидирекционарна агрегација или Композиција)
- Generalization (Генерализација) [11]

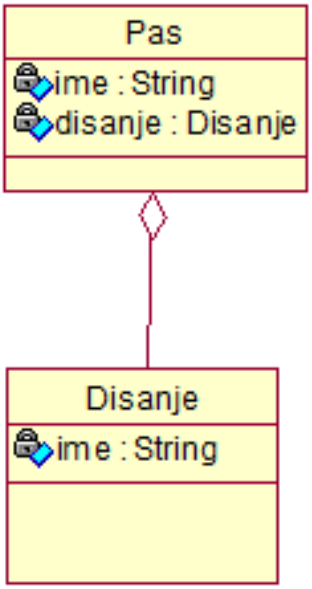
Асоцијација (*Association*)



Слика 6.11. Пример асоцијације

Ако имамо класу које које су у директној вези једна са другом али не морају да атрибуте који су приписани једни другима оваква веза се назива асоцијација. На примеру се може видети да имамо класу Студент, Професор и Школа. У класи Школа се види да су студнет и професор у директној зависности. Зна се да професори имају студенте односно у другачијем случају не би били професори, а овом везом се показује да они не зависе један од другог и ако су повезани на неки начин у класи школа. Кратко речено они су повезани у једној класи али не зависе директно један од другог.

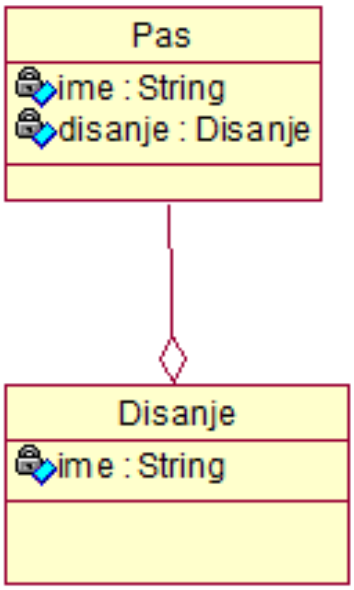
Агрегација (Aggregation)



Слика 6.12. Пример агрегације

У овом примеру се може видети да дисање има агрегациону везу са псом што значи да сваки пас мора да дише на неки начин али ако се погледа даље он припада скупу објеката који се могу објаснити класом дисање. Да би било мало јасније може се рећи да је могуће да се за разне класе паса користи класа дисање што значи да сваки пас има класу дисање али то дисање може да се обавља на различит начин.[7]

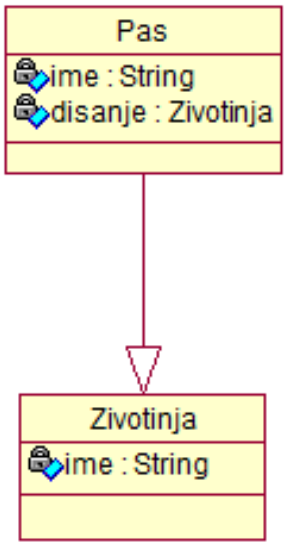
Унидирекционарна(Једносмерна) Агрегације или Композиција (Unidirection Aggregation)



Слика 6.13. Пример унидирекционе агрегације

Ова веза може да се покаже са датог примера изнад и она значи да пас има дисање тј. у класа дисање који ће бити само обичан стринг и налази се као атрибут у класи пас и оваква веза се назива Unidiraciotn Aggregation зато што је дисање саставни део сваке класе пас. Класа дисање не може да постоји без класе пас.

Генерализација (Generalization)



Слика 6.14. Пример генерализације

Ова веза настаје када имамо подкласу креирану од неке друге класе. На овом прмеру се види подкласа пас која је креирана из класе животиње и када се креира подкласа сва поља и методе су дељена између ових класа ако је видљивост постављена као *public, protected or default.*

Кардиналност (Multiplicity)

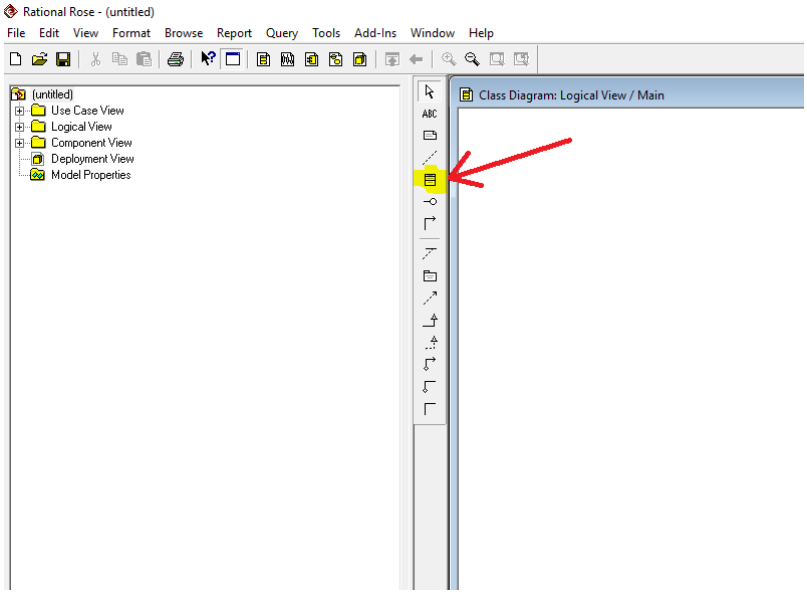
Омогућава да се поставе нумеричка ограничења за везе, а та нумеричка ограничења су:

- 0..1 нула до 1
- n специфичан број
- 0..* нула до више
- 1..* нула до више
- m..n специјалан распон бројева [11]

Појашњење овога и нешто више о томе биће објашњено на примеру дијаграма класа.

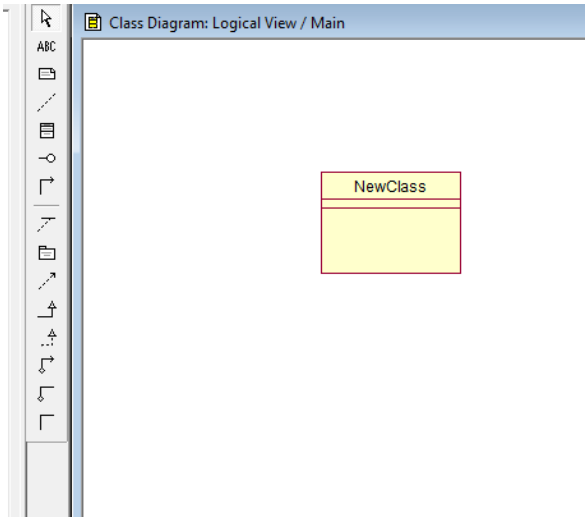
Пример:
Израда дијаграма класа(Class Diagram) у већ поменутом програмском пакету примењен на интернет картици за куповину.

После отварања почетног прозора програма потребно је кликнути на иконицу за нову класу:



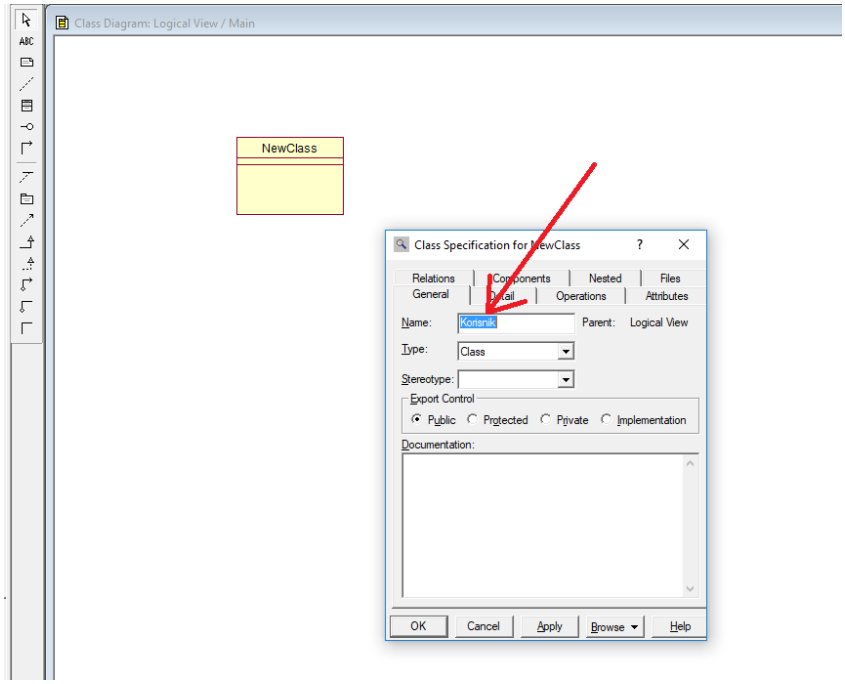
Слика 6.15. Прављење класе

Када се то уради појавиће се прозор односно класа коју је потребно попунити на одговарајући начин.



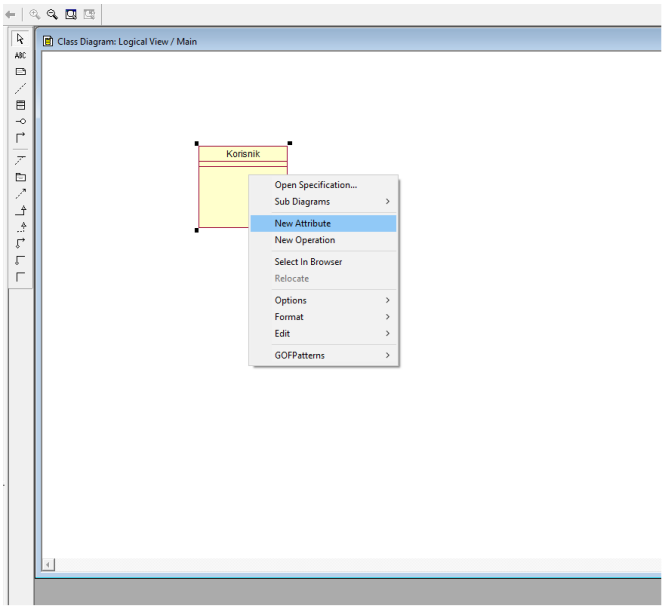
Слика 6.16. Класа

Следеће корак је додељивање имена класи, а то се ради тако што се два пута кликне на класу и у прозору за спецификацију који се отвара и промени се име класе.



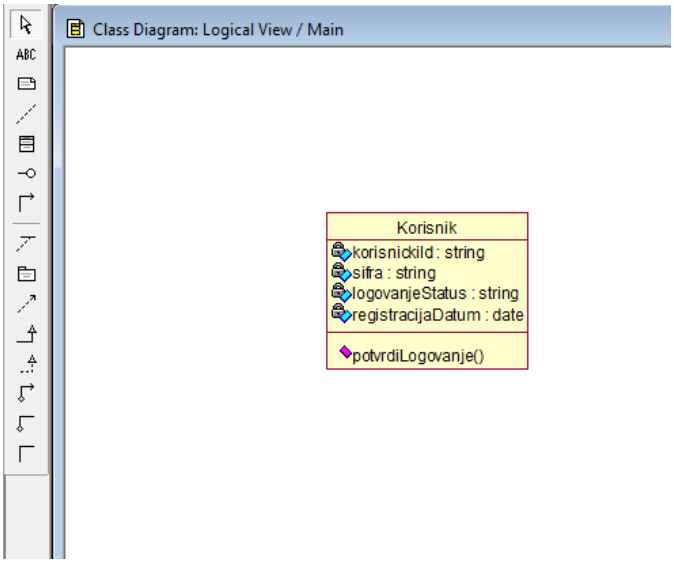
Слика 6.17. Додела назива класе

Када се додели име класи следи додавање атрибута и оператора, а то се врши тако што се притисне десни тастер миша на класу и одабере се поље за додељивање атрибута или оператора.



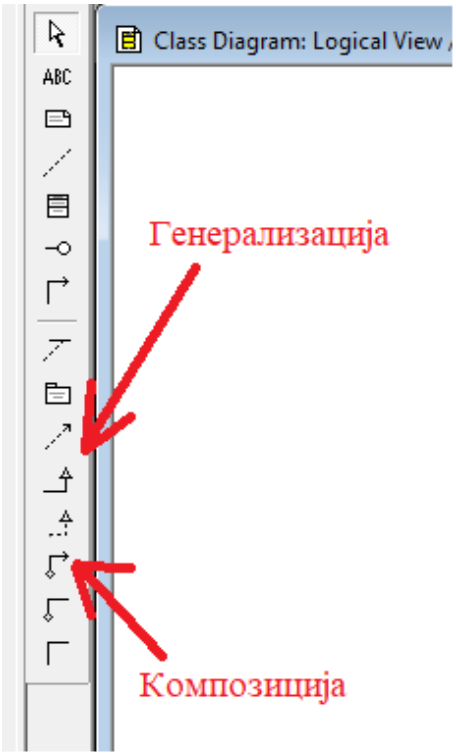
Слика 6.18. Додела атрибура и оператора

Када се унесу потребни атрибути и оператори то би требало овако да изгледа:

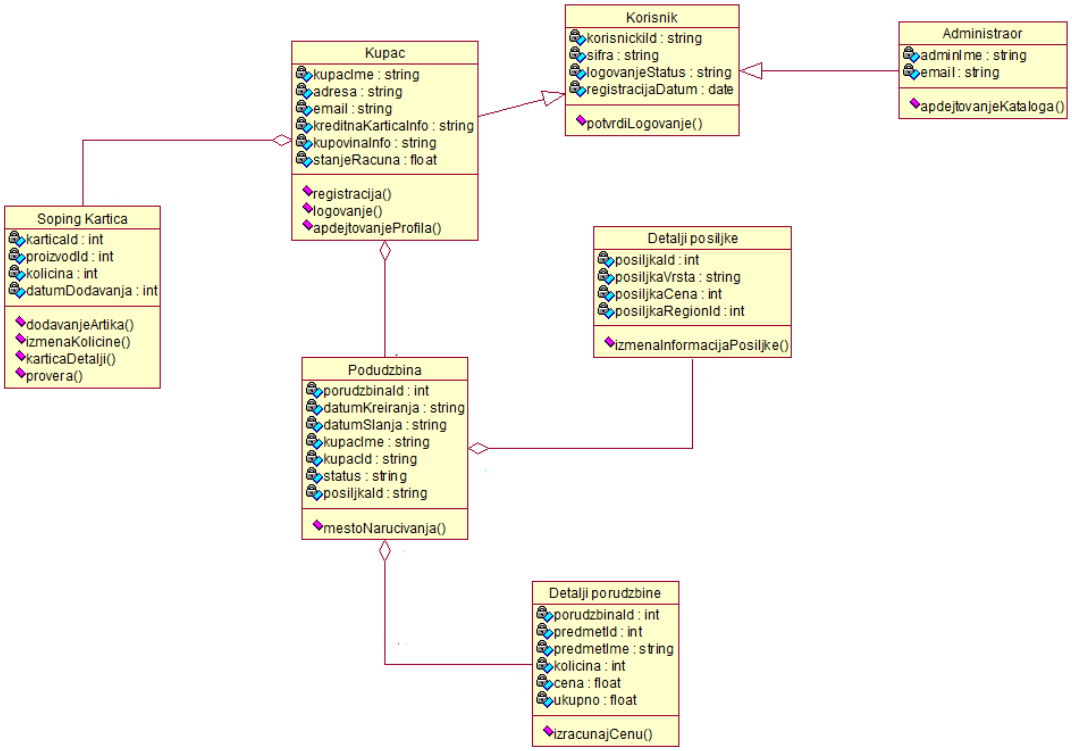


Слика 6.19. Изглед класе

На исти начин се креирају и све остале класе, као и атрибути и оператори унутар тих класа.
Када се креирају све потребне класе са свим својим саставним деловима следи део у коме је потребно да се повежу на неки начин односно класе морају да буду у неким односима. То ће се за овај пример урадити везама генерализација и композиција.



Слика 6.20. Приказ иконица за генерализацију и композицију

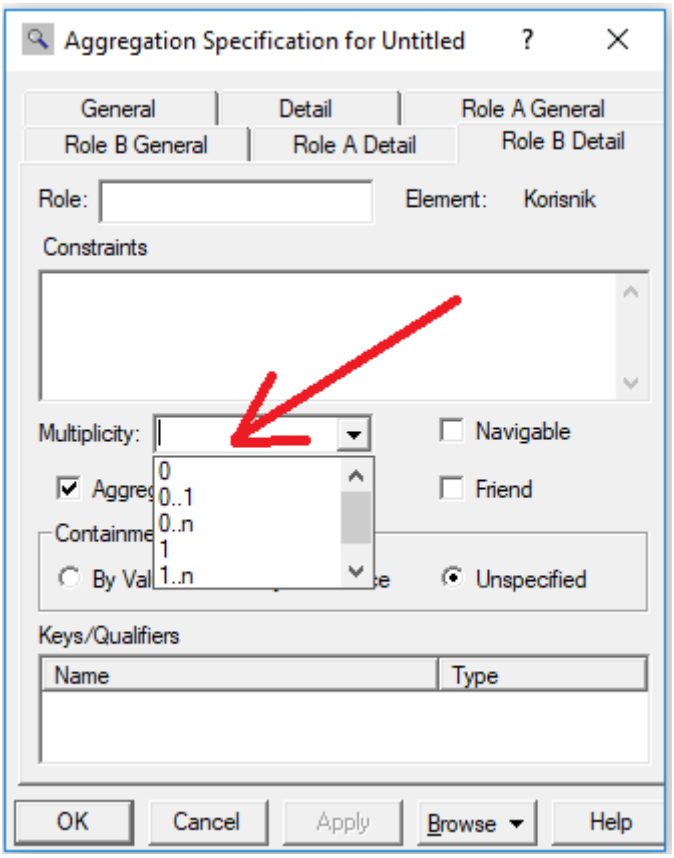


Слика 6.21. Дијаграм класа са везама

Као што се може видети класе купац и администратор су повезани са класом корисник генерализационом везом што значи да су они деца класе корисник и да имају све његове атрибуте и операције али такође имају и своје које их посебно одређују.

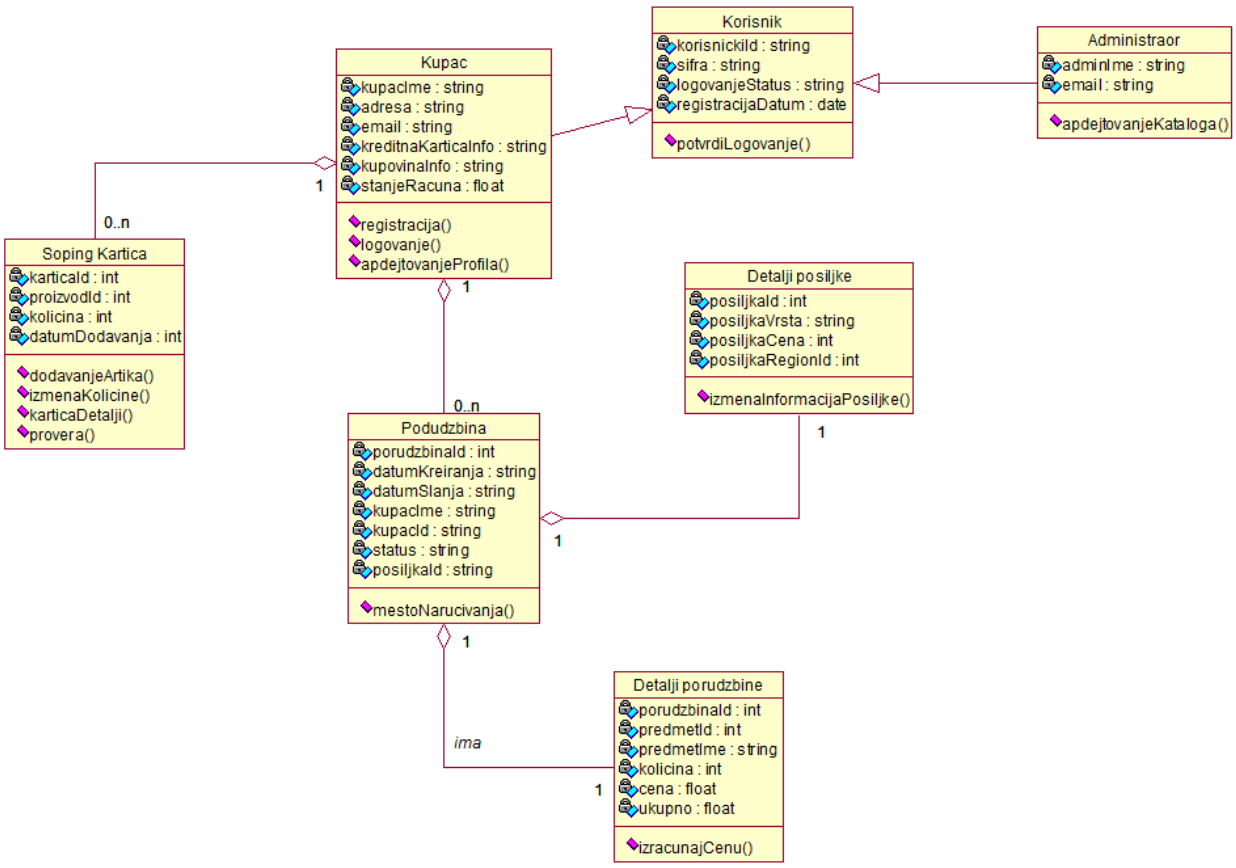
Остале везе представљају композиционе везе што значи да класа шопинг картица не може да постоји без класе купац. Ако се нпр. деси да класа купац буде избрисана онда неће постојати ни остале класе које су са њом композиционо везане, што значи да ове остале класе директно зависе од класе корисник. Исто се може рећи и за остале класе нпр. поруџбеница, ако ње нема онда неће постојати ни класе детаљи поруџбине и детаљи пошиљке зато што оне директно зависе од класе поруџбеница која такође директно зависи од корисника.

Последње што остаје су нумеричка ограничења класа, а она се додељују тако што се десним тастером миша притисне два пута на везу и у прозору за спецификацију из падајућег менија додели одређено ограничење.



Слика 6.22. Приказ додељивања ограничења

Ограничења се постављају за А и Б односно за једну и за другу класу, а та ограничења су она која су већ поменута у претходном делу: 0..1 нула до 1, 0..* нула до више, 1..* нула , m..n специјалан распон бројева, n специфичан број.
Завршен дијаграм класа за интернет картицу као и огранинња може се видети на слици испод.



Слика 6.23. Пример дијагрма класа

Ограничење везе између класе Подуџбина и Купац показује да купац може да има 0 или више поруџбина што значи да купац може да креира налог али не мора да значи да ће да наручи нешто, а може да наручи и више ствари. Иста ова логика важи и за ограничење између класе Шопинг Картица и Купац.

Између класе Поруџбина и Детаљи Поруџбине стоји ограничење 1 до 1, што значи да једна поруџбина може да има само једне детаље производа, а такође само једне детаље прозвода може да има само једну поруџбину. Такође исто важи за за класу Поруџбеница и Детаљи пошиљке.

Закључак

У раду је дато објашњење како су настале базе података и зашто се јавила потреба за њима, као и ко се сматра њеним оснивачем. Прошли смо пут како су се базе података развијале кроз историју. Дато је објашњење неких модела база података, од чега се састоје и чему служе. Објашњено је шта значе ентитети, атрибути, кључеви и везе и зашто су они важни при сваком креирању базе података. Касније је дато објашњење шта значи визуелно моделирање и који се програмски језик користи при овом поступку. Објашњени су неки основи делови програма коришћеног за приказ визуелног моделирања. Дато је кратко објашњење за поједине дијаграма који се могу креирати и дато је детаљно објашњење за креирање дијаграма класа. Као што смо већ рекли визуелно моделирање, односно модели нам помажу да организујемо, визуализујемо, разумемо и правимо сложене системе. Они се користе, а и користиће се, да би нам помогли да се суочимо са изазовима развоја софтвера.

Литература

[1] http://istorija-razvoja-baze-podataka.blogspot.rs/, октобар 2017

[2] https://www.raf.edu.rs/citaliste/istorija/3624-razvoj-baza-podataka, октобар 2017

[3] https://www.google.com/patents/US395782, октобар 2017

[4] Јелена Граовац, Пројектовање базе података
http://poincare.matf.bg.ac.rs/~jgraovac/courses/projbp/2016_2017/projbp_skripta.pdf,
октобар 2017

[5] Роберт Мангер, Базе података
http://jadran.izor.hr/~dadic/EKO/baze-podataka.pdf , октобар 2017

[6] Б. Лазаревић, З. Марјановић, Н. Аничић, С. Бабарогић, Базе података, Београд, 2003

[7] Terry Quatrani, Visual Modeling with Rational Rose 2002 and UML, Addision-Wesley 2003

[8] https://www.techopedia.com/definition/16432/rational-rose, новембар 2017

[9] Милан Ерић, наставни материјал

[10] О.Коцић, С.Мијалковић, П.Вујић, UML, Математички факултет Београд

[11] Anis Yousefi, Rational Rose Tutorial, McMaster University