УНИВЕРЗИТЕТ У БЕОГРАДУ

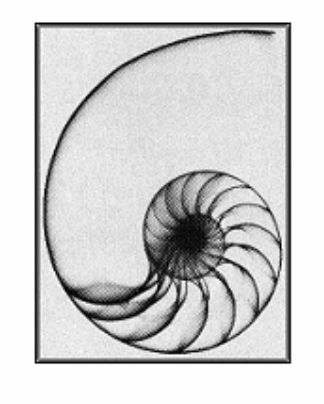
ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА

(Катедра за софтверско инжењерство)

ПРОЈЕКТОВАЊЕ СОФТВЕРА

(СКРИПТА)

Аутор: др Синиша Влајић



Београд - 2015.

Аутор
Др Синиша Влајић, доц.

Наслов
ПРОЈЕКТОВАЊЕ ПРОГРАМА (СКРИПТА)

Издавач
Др Синиша Влајић, Београд

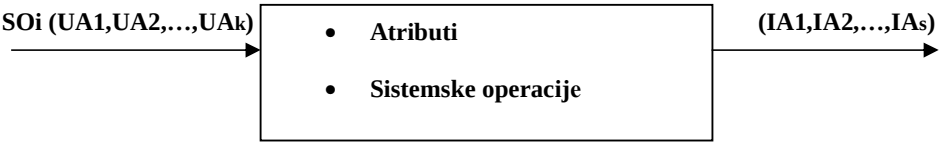
Е-пошта издавача:
vlajic@fon.rs

1. ОСНОВЕ ПРОЈЕКТОВАЊА СОФТВЕРА

1.1 Софтверски систем[1]

Софтверски систем[2] се састоји од **атрибута** и **системских операција**[3]. Атрибути описују **структуру** система, док системске операције описују **понашање** система. Атрибути представљају концепте реалног система који описују статичке карактеристике система (нпр. *Račun, Partner, ..*). Системске операције представљају **основне** (атомске) **функције** система, које се могу користити из окружења система (*нпр. UnesiRačun, PromeniRačun, IzracunajRacun, ...*).
Допустиви улаз у софтверски систем је дефинисан потписом (сигнатуром), који садржи назив системске операције која се позива и скупом улазних аргумената (*нпр. IzracunajRacun(Racun),...*). **Излаз** из софтверског система је представљен преко скупа излазних аргумената (*нпр. Racun, signal, ...*). Излаз се добија као резултат извршења неке од системских операција над атрибутима система.

СОФТВЕРСКИ СИСТЕМ



Atributi = {AT1, AT2,…,ATn}

Sistemske operacije = {SO1,SO2,..,SOm} - Основне функције система

SOi ∈ Sistemske operacije , i = (1,..,m)

UA1,UA2,…,UAk – Улазни аргументи

IA1,IA2,…,IAs – Излазни аргументи

---

[1] Радећи на овом материјалу, често сам долазио у наизглед нерешиве ситуације код повезивање спецификације проблема са њеном реализацијом (имплементацијом). Оно што је правило највећи проблем јесте схватање шта је софтверски систем и које су његове границе. Питање које ми је задавало највише главобоље односило се на кориснички интерфејс: *Да ли је кориснички интерфејс део софтверског система или је он изван система.*
Одговор на наведено питање, и на многа друга око схватања шаренила многих појмова и термина који се користе у софтверском инжењерству, дао ми је проф. Лазаревић Бранислав. Он ми је рекао да је кориснички интерфејс реализација улаза и/или излаза софтверског система. У првом тренутку то ми је изгледало нелогично, али након неког времена, јасна и прецизна аргументација проф. Лазаревића ме је уверила у исправност његових ставова. Наведени став смо представили преко следеће дефиниције:

Деф. Лаз1: Кориснички интерфејс представља реализацију улаза и/или излаза софтверског система.

Наведена дефиниција, колико год једноставно изгледала, је отворила пут да софтверски систем схватимо као систем у правом смислу дефиниције система[РВ]. Надам се да ће системски приступ у схватању развоја софтвера, коначно од софтверског инжењерства направити нешто што ће бити вредно научног а не само технолошког поштовања.

[2] У даљем тексту систем.

[3] ЈПРС користи термин системска операција када жели да нагласи разлику између операција софтверског система и операција које се налазе изван софтверског система.

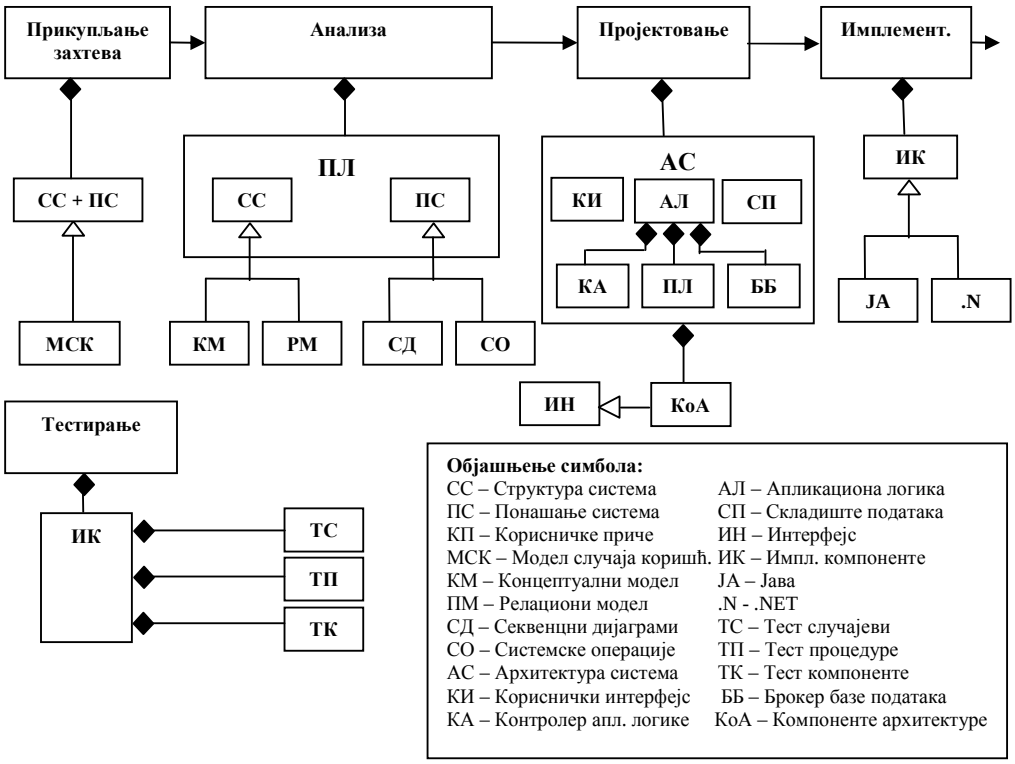
1.2 Контекст пројектовања софтвера

1.2.1 Развој (животни циклус) софтверског система

Развој (животни циклус) софтверског система[4] се састоји из следећих фаза (Слика RASS):

- Прикупљања захтева од корисника
- Анализе
- Пројектовања
- Имплементације
- Тестирања

У фази прикупљања захтева се дефинишу својства и услови које софтверски систем или шире гледајући пројекат треба да задовољи [Larman]. Захтеви се могу поделити као функционални и нефункционални. Функционални захтеви дефинишу захтеване функције система, док нефункционални захтеви дефинишу све остале захтеве. Нефункционални захтеви као што су *употребљивост*, *поузданост*, *перформансе* и *подржаност система* представљају атрибуте квалитета (*quality attributes*) софтверског система. Функционални захтеви се описују преко модела случаја коришћења. Наведени модел садржи елементе структуре и понашања софтверског система. Елементи структуре софтверског система се описују помоћу *именица* и *придева* док се елементи понашања описују помоћу *глагола*. У фази анализе се описује логичка структура и понашање софтверског система односно *пословна логика* софтверског система.



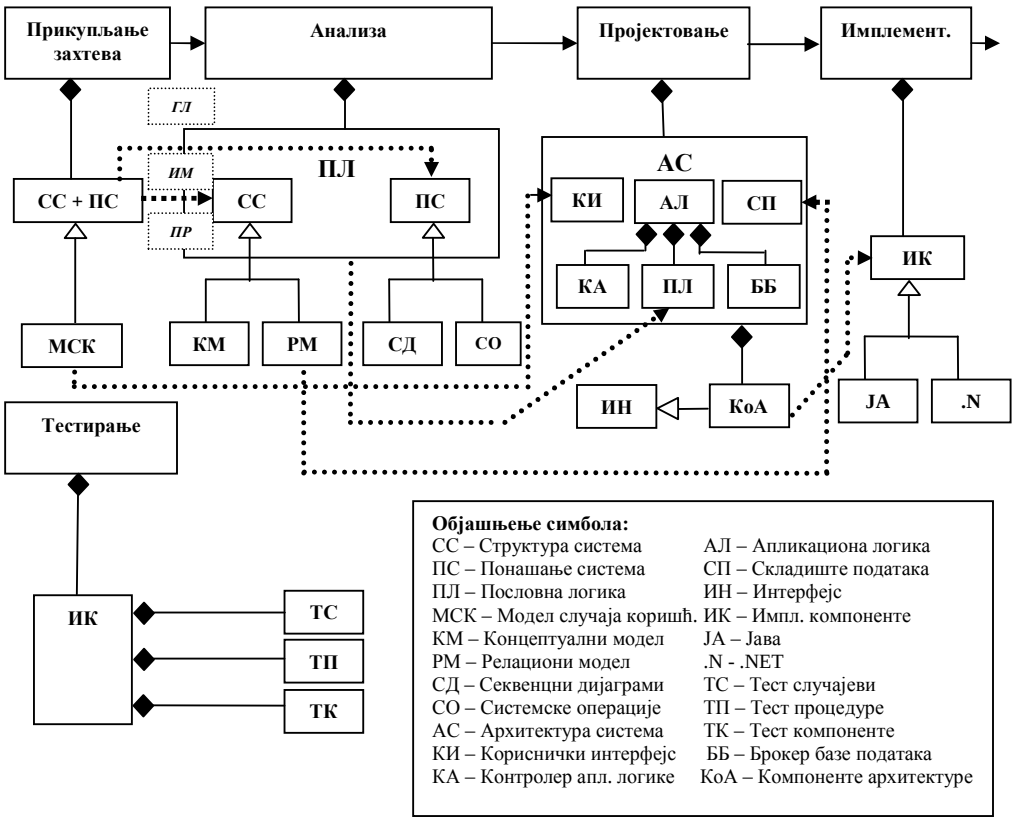
Слика RASS: Развој софтверског система

Структура софтверског система се описује преко *концептуалног (доменског)* и *релационог модела* док се понашање софтверског система описује помоћу *секвенцних дијаграма* и *системских операција*. У фази пројектовања се описује *архитектура* софтверског система која је *тронивојска* и састоји се од: а) *корисничког интерфејса* б) *апликационе логике* и ц) *складишта података*. Апликациона логика се састоји од: б1) *контролера апликационе логике* б2) *пословне логике* и б3) *брокера базе података*. Сваки од наведених нивоа тронивојске

[4] Развој софтверског система је објашњен помоћу поједностављене Ларманове методе развоја софтвера која се користи на ФОН-у, у извођењу наставе на предмету Пројектовања програма и Пројектовање софтвера, од 2003. године.

архитектуре је реализован преко скупа *класа* које представљају *компоненте* архитектуре софтверског система. Компоненте архитектуре су дефинисане преко *интерфејса*. У фази имплементације се праве *имплементационе компоненте* које се реализују у некој од постојећих технологија (*Јава, .NET,...*). У фази тестирања, свака од компоненти се тестира тако што се за сваку од њих праве *тест случајеви*, *тест процедуре* и *тест компоненте*.

Развој софтверског система код Ларманове методе има јасан логички след *(Слика* MERSS*)*. У фази прикупљања захтева елементи структуре и понашања софтверског система се налазе заједно у случајевима коришћења и они су представљени преко именица, глагола и придева. Структура софтверског система (концепти и атрибути концепата), из фазе анализе, се добија на основу именица и придева који су дефинисани у случајевима коришћења. Системске операције, из фазе анализе, се добијају на основу глагола који су дефинисани у случајевима коришћења. Секвенцни дијаграми, из фазе анализе, се добијају на основу сценарија случајева коришћења.



Слика MERSS: Међузависност елемената у развоју софтверског система

У фази пројектовања се прави архитектура софтверског система која је тронивојска (кориснички интерфејс, апликациона логика и складиште података). Кориснички интерфејс је дефинисан преко скупа екранских форми. Сценарија коришћења екранских форми је директно повезан са сценаријима случајева коришћења[5]. Пословна логика, која се добија у фази анализе, се преноси у фазу пројектовања и постаје саставни део апликационе логике. Складиште података се пројектује на основу релационог модела. Имплементационе компоненте, из фазе имплементације, треба да реализују компоненте које су добијене у фази пројектовања. Свака од имплементационих компоненти се тестира у фази тестирања.

[5] Сценарија коришћења екранских форми у суштини представља упуство за крајњег корисника како се користи програм (корисничко упутство).

1.2.2 Однос информационог и софтверског система

Информациони систем (ИС) се прави како би се олакшао рад и управљање неким реалним пословним системом. Пословни систем у најопштијем смислу има своју *структуру* и *понашање*. Структура пословног система се односи на *организациону структуру* преко које се одређују везе и односи између елемената *пословног система*, док се понашање пословног система односи на *пословне процесе* који одређују токове извршења функција пословног система (нпр. функција продаје, набавке,...).

Структура и понашање пословног система се моделирају преко *модела података* и *модела процеса* ИС-а.
Развој ИС-а подразумева прављење модела процеса и модела података пословног система, који требају да буду реализовани у неком технолошком окружењу (оперативни систем, систем за управљање базом података, програмски језик,...).
Модел процеса се описује помоћу *Структурне Систем Анализе (ССА)*[6]. Структурна систем анализа је представљена преко *дијаграма токова података (ДТП)*. На основу дијаграма тока података могуће је направити *речник података* и дати прецизну спецификацију основних (примитивних) процеса система.

Напомена: Структурна систем анализа се ради у фази анализе ИС-а.

Прва фаза у развоју софтверског система, прикупљање захтева, се описује преко *модела случаја коришћења* (СК) који се добија на основу спецификације основних процеса.

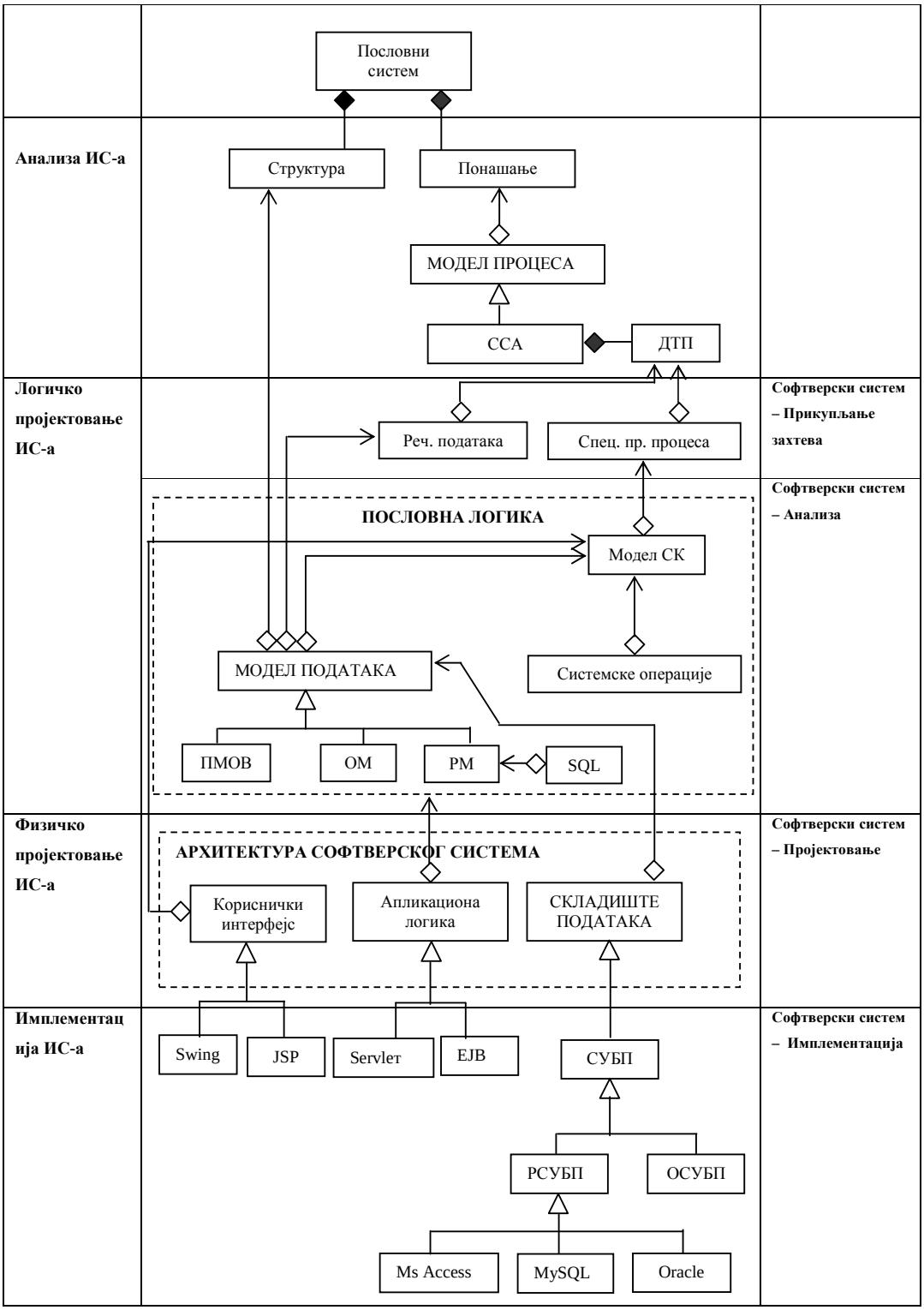
У фази анализе софтверског система, на основу модела СК се одређује *модел података* (структуре) и *модел понашања* софтверског система. Модел података софтверског система се описује помоћу *проширеног модела објекти везе (ПМОВ), релационог модела (РМ), објектног модела (ОМ)*,..., итд. Релациони модел је подржан *SQL* упитним језиком. Релациони модел се може добити на основу ПМОВ-а. Модел понашања софтверског система се описује помоћу *системских операција*. Модел података и модел понашања софтверског система описују *пословну логику* софтверског система.

Напомена: Фаза прикупљања захтева и анализа софтверског система се из перспективе ИС-а називају фаза логичког пројектовања ИС-а.

У фази пројектовања софтверског система се дефинише *тронивојска архитектура* која се састоји од: *корисничког интерфејса, апликационе логике и складишта података.*
Пројектовање сценарија коришћења *екранских форми* корисничког интерфејса се ради на основу СК-а. Кориснички интерфејс се може имплементирати коришћењем разних технологија (нпр. *Swing*, *JSP*,...). Апликациона логика се прави на основу пословне логике софтверског система. Апликациона логика се може реализовати различитим технологијама (Servlet, EJB,...).
У складишту података се чувају подаци. Складиште података може бити реализовано преко система за управљање базом података, системом датотека,...,итд. Системи за управљање базом података (СУБП) могу бити: релациони (РСУБП), објектни (ОСУБП),...,итд. Релациони СУБП су: MS Access, MySQL, Oracle,.., итд.

Напомена: Фаза пројектовања софтверског система се из перспективе ИС-а назива фаза физичког пројектовања ИС-а. Фаза имплементације софтверског система се из перспективе ИС-а назива фаза имплементације ИС-а.

[6] Постоје и други модели процеса који овде неће бити разматрани.

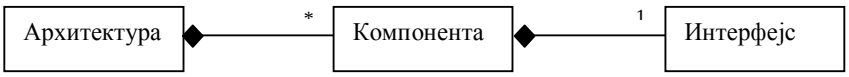


Слика OISS: Однос информационог и софтверског система

1.3 Дефиниција пројектовања софтвера

Пројектовање се у контексту софтверског инжењерства дефинише као:

a) *процес дефинисања архитектуре, компоненти, интерфејса и других особина система или компоненте и*
b) *резултат тог процеса.*



Код објашњења процеса животног циклуса софтвера пројектовање софтвера се састоји од две активности:

- **Пројектовање софтверске архитектуре** (понекад се назива пројектовање највишег нивоа), описује структуру и организацију софтвера на највишем нивоу. На овом нивоу се идентификује различите компоненте.
- **Детаљно пројектовање софтвера** које описује сваку компоненту до нивоа детаљности који је довољан да се она може конструисати.

2. Развој софтверског система
2.1 Прикупљања захтева од корисника

2.1.1 Захтеви (Requirements)
Захтеви представљају својства и услове које систем или шире гледајући пројекат мора да задовољи [Ларман]. Постоје различити типови захтева које систем мора да задовољи и они су категоризовани према **FURPS+** (**Functional** - *Функционалност*, **Usability** - *Употребљивост*, **Reliability** - *Поузданост*, **Performance** - *Перформансе*, **Supportability** - *Подрживост*) моделу:

- *Функционалност* представља способност (**capabilities**) система да обезбеди захтеване функције (понашање система). Заштита (**security**) система представља једну од основних функција коју систем треба да обезбеди.
- *Употребљивост* представља способност система да се може једноставно користити. То се постиже помоћу разних упутстава и документције који описују начин његовог коришћења.
- *Поузданост* представља способност система да може успешно обрадити проблем (**failure**) који се дешава у току извршења система. У том смислу систем мора да обезбеди начин опоравка (**recoverability**) података у случају насилног прекида рада система. Такође систем треба да омогући предвиђање (**predictability**) могућих понашања система.
- *Перформансе* система се односе на време одзива (**response time**) захтеваних функција, пропусну моћ (**throughput**) мреже кроз коју пролазе подаци, тачност (**accuracy**) извршења функција, могућност коришћења односно расположивост (**availability**) функција система и начин коришћење расположивих ресурса(**resource usage**) система.
- *Подрживост* система се односи на лакоћу његовог прилагођавања (**adaptability**) и одржавања (**maintainability**), интернационализацију (**internationalization**) у смислу његове прилагодљивости различитим знаковним системима који се користе у свету и начину конфигурисања (**configurability**) система.

Захтеви се често категоризују као функционални и не функционални захтеви. Функционални захтеви дефинишу захтеване функције система, док не функционални захтеви дефинишу све остале захтеве. У том смислу не функционални захтеви (употребљивост, поузданост, перформансе и подрживост система) представљају атрибуте квалитета (**quality attributes**) софтверског система.

У ФУРПС+ моделу знак ‘**+**’ указује на помоћне захтеве који се односе на:

- *Имплементацију* (**Implementation**) система – до којих граница се могу користити расположиви ресурси (**resource limitations**). Који се програмски језици (**programming languages**) и алати (**tools**) могу користити. Поред тога имплемантациони захтеви се односе и на хардвер (**hardware**) који ће се користити.
- *Интерфејс* (**Interface**) система – ограничења која постоје у комуникацији система са његовим окружењем (екстерним системима).
- *Операције* (**Operations**) система – управљање системом и његовим операцијама.
- *Паковање* (**Packaging**) система – начин физичког организовања система у пакете, који представљају управљиве јединице система.
- *Легалност* (**Legal**) – могућност употребе система у смислу његове легалности (лиценце и права коришћења система).

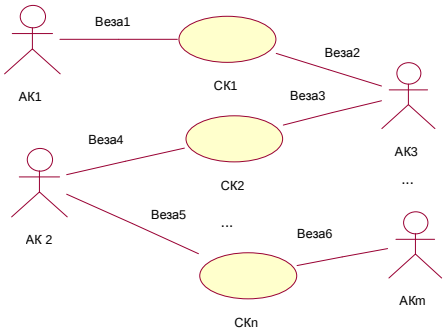
У даљем тексту ми ћемо главни нагласак ставити на разматрање функционалних захтева, од њиховог прикупљања до имплементације. Сматрамо да истовремено објашњење функционалних и не-функционалних захтева, које прати студијски пример, може доста да усложи схватање суштине развоја једног софтверског система из угла његове основне функционалности. У студијском примеру ми ћемо увести неки од не-функционалних захтева који су директно повезани са функционалношћу система (*поузданост и подрживост система*) и неке од помоћних захтева (*имплементација, операције и паковање система*) Остали не-функционални и

помоћни захтеви неће бити разматрани, будући да су они у највећој мери ортогонално постављени у односу на функције система. То значи да они не утичу на схватање и објашњење развоја функција система.

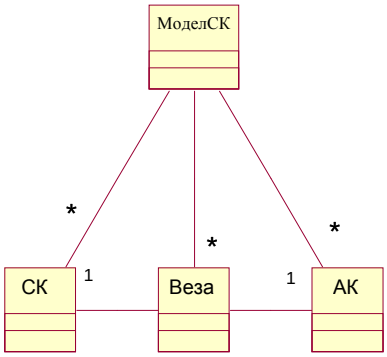
2.1.2 Опис захтева помоћу модела случаја коришћења

Захтеви се код Лармана описују помоћу UML *Модела Случаја Коришћења* (**Use-Case Model**).

Деф. ЗА1: Модел СК се састоји од скупа случаја коришћења (СК), актора (АК) и веза између случаја коришћења и актора.

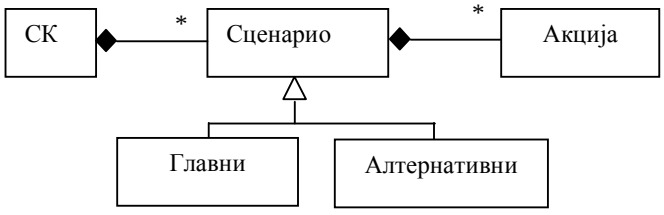


Модел СК се може представити са:



Модел СК је везан за више а) СК, б) веза између СК и АК и ц) АК. Један СК може да има више веза са АК. Један АК може да има више веза са СК. Једна веза се односи на један пар СК-АК.

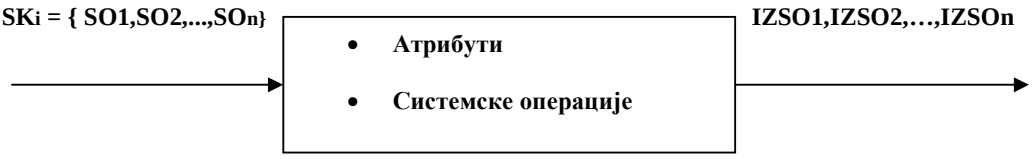
Деф. ЗА2: Случај коришћења описује скуп **сценарија** (**use-case појављивања**), односно скуп жељених коришћења система од стране актора.



Сценарио описује једно жељено коришћење система од стране актора. Случај коришћења има један главни и више алтернативних сценарија. Сценарио је описан преко: а) секценце акција и б) интеракција између актора и система. СК се састоји из главног и алтернативних сценарија.

У току интеракције између актора и софтверског система, актор позива системске операција софтверског система. То значи да се у току неког (једног) сценариа *Ski* случаја коришћења позива једна или више системских операција (*SO1*, *SO2*,...,*SOn*).

СОФТВЕРСКИ СИСТЕМ



Као резултат извршења системске операције над атрибутима софтверског система се добијају излазни резултати IZSO1, IZSO2,...,IZSOn.

Системске операције = {SO1,SO2,..,SOn} - Основне функције система (атомске функције)

Случајеви коришћења = {SK1,SK2,..,SKp} - Жељене функције система (молекулске функције)

SKi ∈ Slučajevi korišćenja, i =(1..p)

СК, односно сценарија СК дефинишу жељене функције система. Жељене функције система, када се извршавају, позивају по одређеном редоследу основне функције система.

Једну акцију сценариа изводи или актор или систем. У том смислу:
Актор изводи једну од три врсте акција:
а) **А**ктор **П**рипрема **У**лаз (Улазне Аргументе) за **С**истемску **О**перацију (**АПУСО**).
б) **А**ктор **П**озива систем да изврши **С**истемску **О**перацију (**АПСО**).
ц) **А**ктор извршава **НеС**истемску **О**перацију (**АНСО**).

Систем изводи две акције у континуитету:
а) Систем извршава **С**истемску **О**перацију(**СО**):
б) Резултат системске операције (Излазни аргументи (**ИА**)) се прослеђује до актора.

Деф. ЗА3: Актор (учесник) представља спољног корисника система. Он поставља захтев систему да изврши једну или више системских операција, по унапред дефинисаном сценарију[7]. Систем одговара на постављени захтев актора, шаљући му вредност излазних аргумената као резултат извршења операција. Актор се обично дефинише као неко или нешто (нпр: људи, рачунарски системи или организациона јединица) што има понашање.

Правила код прикупљања захтева:

Правило ЗА1 (Независност сценарија СК): Сценарија СК не треба да буду у међусобној интеракцији. Она се требају дефинисати као атомска, у смислу да се извршавају у потпуности самостално. На тај начин се олакшава њихов развој и одржавање[JPRS][8].

Правило ЗА2 (Glass' low) [EA1]**:** Недостаци код дефинисања захтева су основни разлог могућег неуспеха у развоју пројекта (програма).

Правило ЗА3 (Boehm's first low) [EA1]**:** Уколико се не уоче грешке у току дефинисања захтева, исте се веома тешко могу уклонити у каснијим фазама развоја програма.

[7] Захтев за извршење једне или више системских операција се не одиграва континуално него дискретно. То значи да корисник интерактивно позива једну по једну системску операцију, у дискретним временским интервалима. Из наведеног може да се закључи да сценарио описује интерактивно коришћење софтверског система.

[8] У разговору са проф. Братиславом Петровићем рекао сам да независност сценарија, повећава редудансу у опису понашања система али истовремено знатно олакшава одржавање система. Рекао сам да су редуданса и одржавање обрнуто сразмерни. Проф. Петровић је рекао да је њихов производ вероватно неки коефицијент. ***Било би веома интересантно када би неко успео да формално објасни тај однос и да пронађе наведени коефицијент.***

Правило ЗА4 (Boehm's second low) [EA1]**:** Прављење прототипова значајно смањује могуће грешке код дефинисања захтева и његовог развоја, нарочито код дефинисања корисничког интерфејса.

Начин представљања модела СК

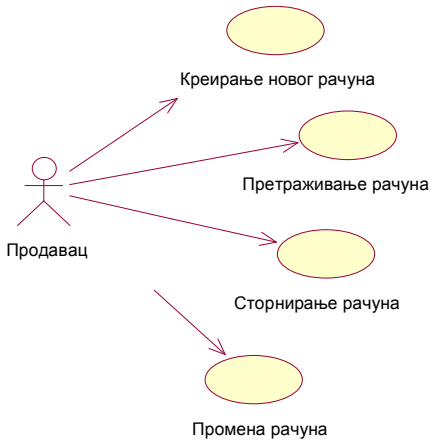
СК се у почетним фазама развоја софтвера представљају текстуално док се касније они представљају преко секвенцних дијаграма, дијаграма сарадње, дијаграма прелаза стања или дијаграма активности.
Тексуални опис СК има следећу структуру:

- Назив СК.
- Акторе СК.
- Учеснике СК.
- Предуслови који морају бити задовољени да би СК почео да се извршава.
- Основни сценарио извршења СК.
- Постуслови који морају бити задовољени да би се потврдило да је СК успешно извршен.
- Алтернативна сценарија извршења СК.
- Специјални захтеви.
- Технолошки захтеви.
- Отворена питања.

У нашем примеру имамо следеће СК-а:

1. *Креирање новог рачуна.*
2. *Претраживање рачуна.*
3. *Сторнирање рачуна.*
4. *Промена рачуна.*

Наведене СК користи Продавац (актор).
Модел СК се може представити преко следећег дијаграма СК:



СК1: Случај коришћења – Креирање новог рачуна

Назив СК
Креирање новог рачуна

Актори СК
Продавац

Учесници СК
Продавац и систем (програм)

Предуслов: Систем је укључен и продавац је улогован под својом шифром. Систем приказује форму за рад са рачуном.

Основни сценарио СК

1. Продавац **позива** систем да креира нови рачун. (АПСО)
2. Систем **креира** нови рачун. (СО)
3. Систем **приказује** продавцу нови рачун и поруку: "Систем је креирао нови рачун". (ИА)
4. Продавац **уноси** податке у нови рачун. (АПУСО)
5. Продавац **контролише** да ли је коректно унео податке у нови рачун. (АНСО)
6. Продавац **позива** систем да запамти податке о рачуну. (АПСО)
7. Систем **памти** податке о рачуну. (СО)
8. Систем **приказује** продавцу запамћени рачун и поруку: "Систем је запамтио рачун. (ИА)
9. Продавац **позива** систем да обради рачун. (АПСО)
10. Систем **обрађује** рачун.(СО)
11. Систем **приказује** продавцу обрађен рачун и поруку: "Систем је обрадио рачун".(ИА)

Алтернативна сценарија

3.1 Уколико систем не може да креира рачун он приказује продавцу поруку: "Систем не може да креира нови рачун". Прекида се извршење сценариа. (ИА)
8.1 Уколико систем не може да запамти податке о рачуну он приказује продавцу поруку "Систем не може да запамти рачун". Прекида се извршење сценариа. (ИА)
11.1 Уколико систем не може да обради рачун он приказује продавцу поруку: "Систем не може да обради рачун". (ИА)

СК2: Случај коришћења – Претраживање рачуна

Назив СК
Претраживање рачуна

Актори СК
Корисник

Учесници СК
Корисник и систем (програм)

Предуслов: Систем је укључен и корисник је улогован под својом шифром. Систем приказује форму за рад са рачуном.

Основни сценарио СК

1. Корисник **уноси** вредност по којој претражује рачун. (АПУСО)
2. Корисник **позива** систем да нађе рачун по задатој вредности. (АПСО)
3. Систем **тражи** рачун по задатој вредности. (СО)
4. Систем приказује кориснику податке о рачуну и поруку: "Систем је нашао рачун по задатој вредности". (ИА)

Алтернативна сценарија

4.1 Уколико систем не може да нађе рачун он приказује кориснику поруку: "Систем не може да нађе рачун по задатој вредности". (ИА)

СК3: Случај коришћења – Сторнирање рачуна

Назив СК
Сторнирање рачуна

Актори СК
Корисник

Учесници СК
Корисник и систем (програм)

Предуслов: Систем је укључен и корисник је улогован под својом шифром. Систем приказује форму за рад са рачуном.

Основни сценарио СК

1. Корисник **уноси** вредност по којој претражује рачун. (АПУСО)
2. Корисник **позива** систем да нађе рачун по задатој вредности. (АПСО)
3. Систем **тражи** рачун по задатој вредности. (СО)
4. Систем приказује кориснику рачун и поруку: "Систем је нашао рачун по задатој вредности". (ИА)
5. Корисник **позива** систем да сторнира задати рачун. (АПСО)
6. Систем **сторнира** рачун. (СО)
7. Систем **приказује** кориснику сторниран рачун и поруку: "Систем је сторнирао рачун ". (ИА)

Алтернативна сценарија

4.1 Уколико систем не може да нађе рачун он приказује кориснику поруку: "Систем не може да нађе рачун по задатој вредности". Прекида се извршење сценариа. (ИА)
7.1 Уколико систем не може да сторнира рачун он приказује кориснику поруку: "Систем не може да сторнира рачун".

СК4: Случај коришћења – Промена рачуна

Назив СК
Промена рачуна

Актори СК
Продавац

Учесници СК
Продавац и систем (програм)

Предуслов: Систем је укључен и продавац је улогован под својом шифром. Систем приказује форму за рад са рачуном.

Основни сценарио СК

1. Продавац **уноси** вредност по којој претражује рачун. (АПУСО)
2. Продавац **позива** систем да нађе рачун по задатој вредности. (АПСО)
3. Систем **тражи** рачун по задатој вредности. (СО)
4. Систем приказује продавцу рачун и поруку: "Систем је нашао рачун по задатој вредности". (ИА)
5. Продавац **уноси (мења)** податке о рачуну. (АПУСО)
6. Продавац **контролише** да ли је коректно унео податке о рачуну. (АНСО)
7. Продавац **позива** систем да запамти податке о рачуну. (АПСО)
8. Систем **памти** податке о рачуну. (СО)
9. Систем **приказује** продавцу поруку: "Систем је запамтио рачун." (ИА)
10. Продавац **позива** систем да обради рачун. (АПСО)
11. Систем **обрађује** рачун.(СО)
12. Систем **приказује** продавцу обрађен рачун и поруку: "Систем је обрадио рачун.(ИА)

Алтернативна сценарија

4.1 Уколико систем не може да нађе рачун он приказује продавцу поруку: "Систем не може да нађе рачун по задатој вредности". Прекида се извршење сценариа. (ИА)
9.1 Уколико систем не може да запамти податке о рачуну он приказује продавцу поруку "Систем не може да запамти рачун". Прекида се извршење сценариа. (ИА)
12.1 Уколико систем не може да обради рачун он приказује продавцу поруку: "Систем не може да обради рачун". (ИА)

2.2 Анализа

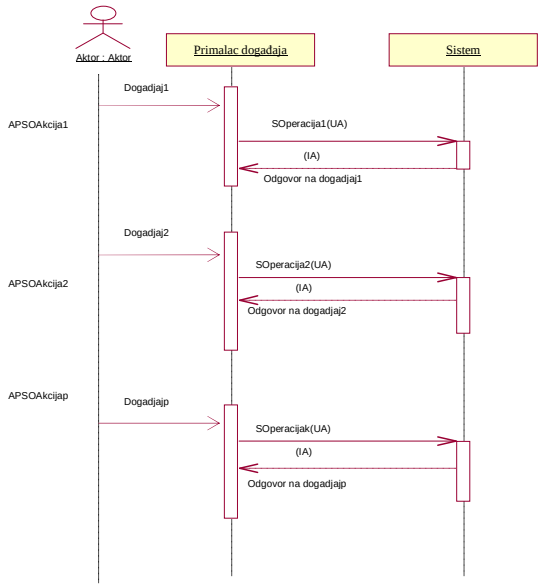
Фаза анализе описује логичку структуру и понашање софтв. система (пословну логику софт. система). Понашање софтверског система је описано помоћу системских дијаграма секвенци и преко системских операција. Структура софтерског система се описује помоћу концептуалног и релационог модела.

2.2.1 ПОНАШАЊЕ СОФТ. СИСТЕМА - СИСТ. ДИЈАГРАМИ СЕКВЕНЦИ

Понашање система се може описати преко УМЛ-ових **секвенцних дијаграма** [Larman]**,** односно преко **дијаграма сарадње** [JPRS]**.**

Деф. АН1: Системски дијаграм секвенци приказује, за издвојени сценарио СК, догађаје у одређеном редоследу, који успостављају интеракцију између актора и софтверског система.

Деф. АН2: Догађај који **направи** актор је побуда за позив системске операције. То значи да актор не позива системску операцију непосредно већ то чини преко посредника (примаоца догађаја). Позив системске операције указује на интеракцију између актора и система. За догађај који представља побуду за позив СО се често каже да је то **системски догађај.**



Догађаје праве актори (*нпр. клик на дугме, које се налази на екранској форми*), у оквиру APSO акција, над примаоцем догађаја (*нпр. дугме*). Прималац догађаја прихвата догађај и позива системску операцију (*нпр. дугме прима клик (догађај) и покреће методу која позива системску операцију*) која се налази на страни система. Након извршења системске операције систем враћа неки резултат као одговор на догађај (*то може да буде сигнал о успешности извршења операције и/или неки податак*).

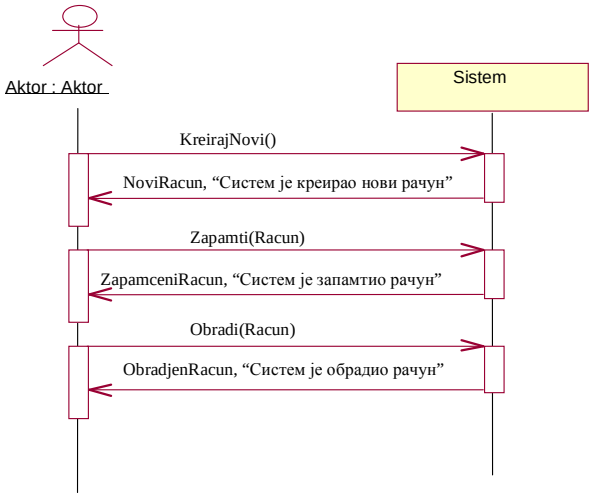
Као резултат анализе сценарија СК добијају се захтеви за извршење системских операција. За сваку системску операцију се праве уговори(контракти).

За сваки СК, прецизније речено за сваки сценарију СК, праве се системски дијаграми секвенци и то само за АПСО и ИА акције сценарија.

ДС1: ***Дијаграми секвенци случаја коришћења – Креирање новог рачуна***

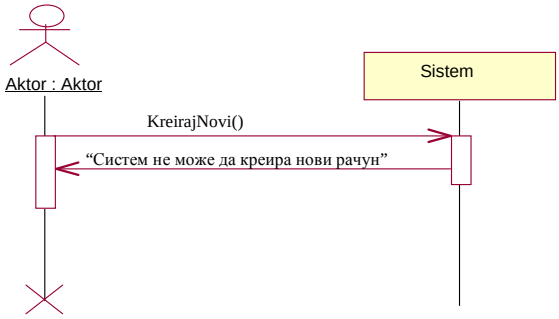
1. Продавац **позива** систем да креира нови рачун. (АПСО)
2. Систем **приказује** продавцу нови рачун и поруку: "Систем је креирао нови рачун". (ИА)
3. Продавац **позива** систем да запамти податке о рачуну. (АПСО)
4. Систем **приказује** продавцу запамћени рачун и поруку: "Систем је запамтио рачун. (ИА)
5. Продавац **позива** систем да обради рачун. (АПСО)
6. Систем **приказује** продавцу обрађен рачун и поруку: "Систем је обрадио рачун".(ИА)

Из наведеног може да се закључи да се на системском дијаграму секвенци не виде АПУСО, СО и АНСО акције.

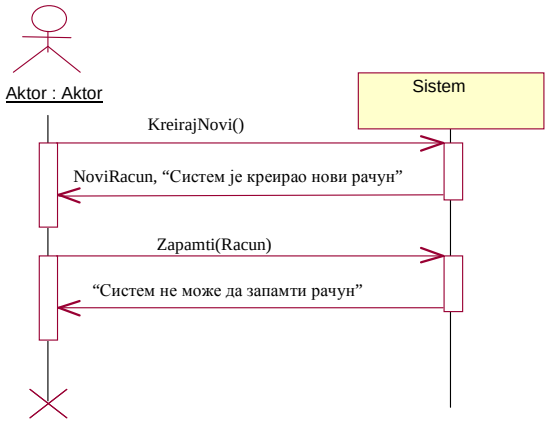


Алтернативна сценарија

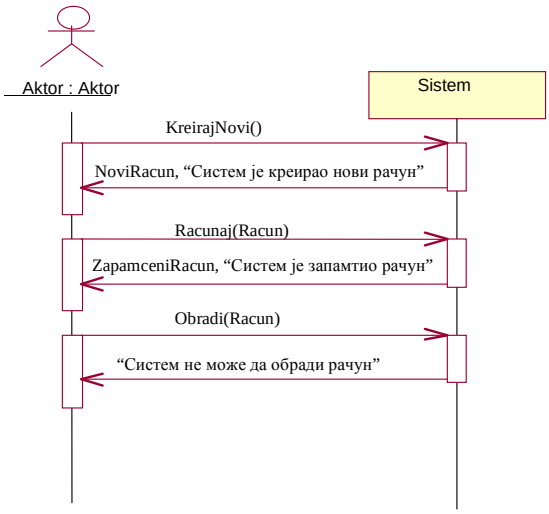
2.1 Уколико систем не може да креира рачун он приказује продавцу поруку: "Систем не може да креира нови рачун". Прекида се извршење сценариа. (ИА)



4.1 Уколико систем не може да запамти податке о рачуну он приказује продавцу поруку "Систем не може да запамти рачун". Прекида се извршење сценариа. (ИА)



6.1 Уколико систем не може да обради рачун он приказује продавцу поруку: "Систем не може да обради рачун". (ИА)

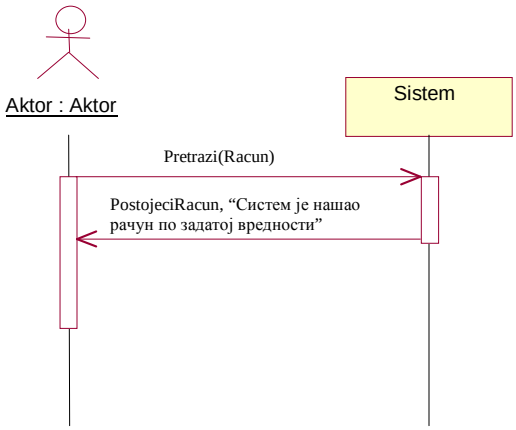


Са наведених секвенцних дијаграма уочавају се 3 системске операције које треба пројектовати:

1. *signal* **KreirajNovi** (*Racun*);
2. *signal* **Zapamti**(*Racun*);
3. *signal* **Obradi**(*Racun*);

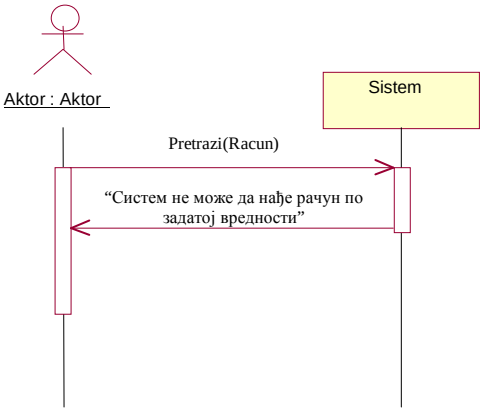
ДС2: *Дијаграми секвенци случаја коришћења – Претраживање рачуна*

1. Корисник **позива** систем да нађе рачун по задатој вредности. (АПСО)
2. Систем приказује кориснику податке о рачуну и поруку: "Систем је нашао рачун по задатој вредности". (ИА)



Алтернативна сценарија

2.1 Уколико систем не може да нађе рачун он приказује кориснику поруку: "Систем не може да нађе рачун по задатој вредности". (ИА)



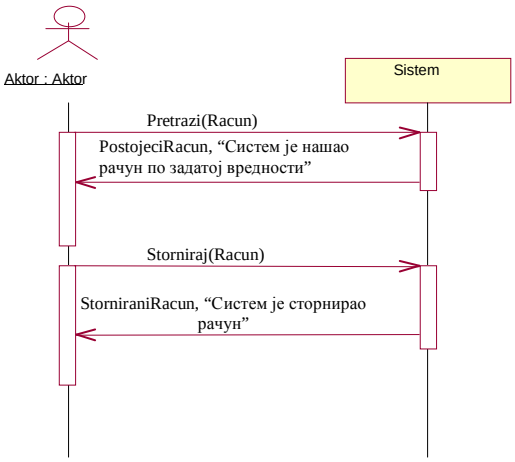
Са наведених секвенцних дијаграма уочава се још једна системска операција коју треба пројектовати:

1. signal **Pretrazi** (*Racun*);

ДС3: ***Дијаграми секвенци случаја коришћења – Сторнирање рачуна***

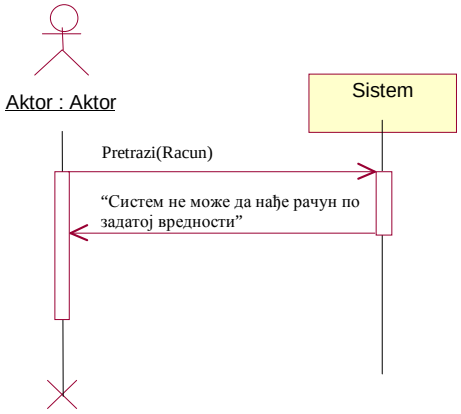
Основни сценарио СК

1. Корисник **позива** систем да нађе рачун по задатој вредности. (АПСО)
2. Систем приказује кориснику рачун и поруку: "Систем је нашао рачун по задатој вредности". (ИА)
3. Корисник **позива** систем да сторнира задати рачун. (АПСО)
4. Систем **приказује** кориснику сторниран рачун и поруку: "Систем је сторнирао рачун ". (ИА)

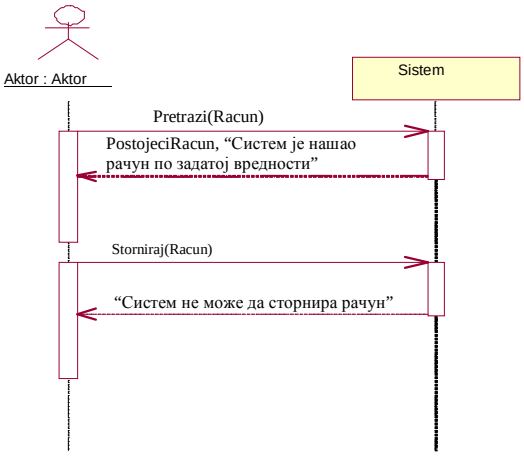


Алтернативна сценарија

2.1 Уколико систем не може да нађе рачун он приказује кориснику поруку: "Систем не може да нађе рачун по задатој вредности". Прекида се извршење сценариа. (ИА)



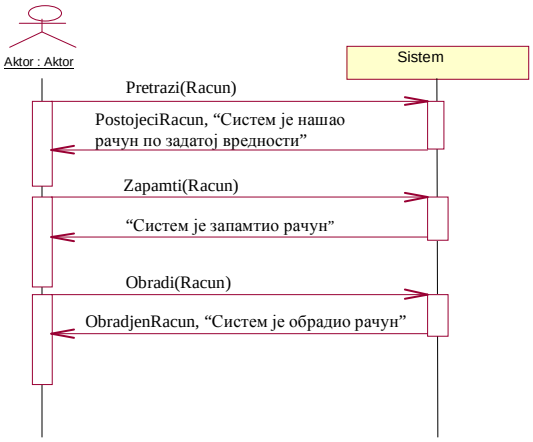
4.1 Уколико систем не може да сторнира рачун он приказује кориснику поруку: "Систем не може да сторнира рачун".



Са наведених секвенцних дијаграма уочава се још једна системска операција коју треба пројектовати:
1. signal **Storniraj** (*Racun*);

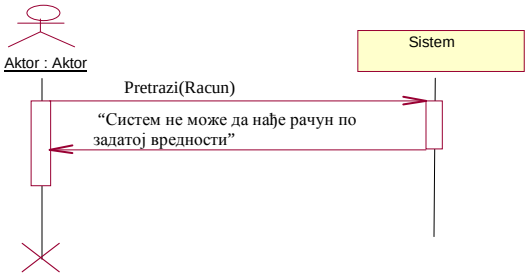
ДС4: *Дијаграми секвенци случаја коришћења – Промена рачуна*

1. Продавац **позива** систем да нађе рачун по задатој вредности. (АПСО)
2. Систем приказује продавцу рачун и поруку: "Систем је нашао рачун по задатој вредности". (ИА)
3. Продавац **позива** систем да запамти податке о рачуну. (АПСО)
4. Систем **приказује** продавцу поруку: "Систем је запамтио рачун." (ИА)
5. Продавац **позива** систем да обради рачун. (АПСО)
6. Систем **приказује** продавцу обрађен рачун и поруку: "Систем је обрадио рачун.(ИА)

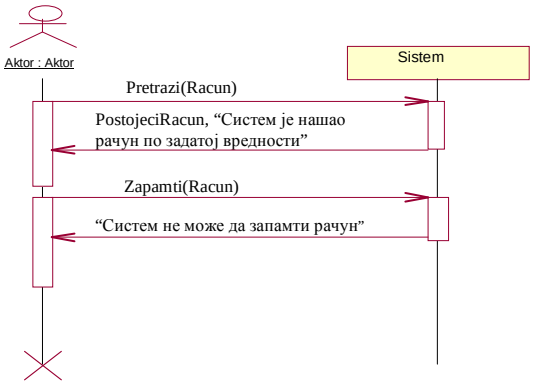


Алтернативна сценарија

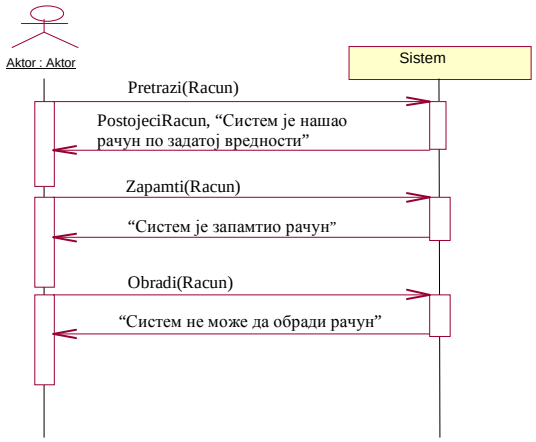
2.1 Уколико систем не може да нађе рачун он приказује продавцу поруку: "Систем не може да нађе рачун по задатој вредности". Прекида се извршење сценариа. (ИА)



4.1 Уколико систем не може да запамти податке о рачуну он приказује продавцу поруку "Систем не може да запамти рачун". Прекида се извршење сценариа. (ИА)



6.1 Уколико систем не може да обради рачун он приказује продавцу поруку: “Систем не може да обради рачун”. (ИА)



Као резултат анализе сценарија добијено је укупно 5 системских операција које треба пројектовати:

1. *signal* **KreirajNovi** (*Racun*);
2. *signal* **Zapamti**(*Racun*);
3. *signal* **Obradi**(*Racun*);
4. *signal* **Pretrazi** (*Racun*);
5. *signal* **Storniraj** (*Racun*);

2.2.2 ДЕФИНИСАЊЕ УГОВОРА О СИСТЕМСКИМ ОПЕРАЦИЈАМА

За сваку од уочених системских операција праве се уговори.

Деф. АН3: Системска операција описује понашање софтверског система. Системска операција има свој потпис, који садржи име методе и опционо улазне и/или излазне аргументе.

Деф. АН4: Уговори се праве за системске операције и они описују њено понашање. Уговори описују шта операција треба да ради, без објашњења како ће то да ради. Један уговор је везан за једну системску операцију.
Уговори се састоје из следећих секција:

- **Операција –** име операције и њени улазни и излазни аргументи
- **Веза са СК –** имена СК у којима се позива системска операција
- **Предуслов** – пре извршења системске операције морају бити задовољени одређени предуслови (систем мора бити у одговарајућем стању).
- **Постуслови –** после извршења системске операције у систему морају бити задовољени одређени постуслови (систем мора бити у одговарајућем стању или се поништава резултат операције).

Деф. АН5: Постуслови СО указују на то шта треба да се деси (ефекти извршења СО), након извршења СО, а не како ће то да се деси.
Постуслови се изражавају у прошлом времену, како би се нагласило да је објекат дошао у ново стање, а не да ће доћи у ново стање.Нпр. *Ставка рачуна је креирана*. Не би било добро да се напише: *Креирање ставке рачуна.*

Деф. АН6: Предуслови СО указују на то шта је требало да се деси, како би СО могла да се изврши, а не како се то десило.

1. Уговор UG1: *KreirajNovi*

Операција: KreirajNovi (*Racun*):signal;
Веза са СК: СКП31
Предуслови: *Вредносна и структурна ограничења над објектом Racun морају бити задовољена.*
Постуслови: *Направљен је нови рачун.*

2. Уговор UG2: *Zapamti*

Операција: Zapamti(*Racun*):signal;
Веза са СК: *СК1, СК4*
Предуслови: *Ако је рачун обрађен или сторниран не може се извршити системска операција. Вредносна и структурна ограничења над објектом Racun морају бити задовољена.*

Постуслови:

- *Израчуната је вредност сваке од ставки рачуна.*
- *Израчуната је укупна вредност рачуна.*

3. Уговор UG3: *Obradi*

Операција: Obradi(*Racun*):signal;
Веза са СК: *СК1, СК4*
Предуслови: *Ако је рачун обрађен или сторниран не може се извршити системска операција. Вредносна и структурна ограничења над објектом Racun морају бити задовољена.*
Постуслови:

- *Израчуната је вредност сваке од ставки рачуна.*
- *Израчуната је укупна вредност рачуна.*
- *Рачун је обрађен.*

4. Уговор UG4: *Pretraži*

Операција: Pretraži (*Racun*):signal;
Веза са СК: *СК2, СК3, СК4*
Предуслови:
Постуслови:

5. Уговор UG5: *Storniraj*

Операција: Storniraj(*Racun*):signal;
Веза са СК: *СК3*
Предуслови: *Ако је рачун сторниран не може се извршити системска операција.*
Постуслови: *Рачун је сторниран*

2.2.3 ОГРАНИЧЕЊА ПРИ ИЗВРШЕЊУ СИСТЕМСКИХ ОПЕРАЦИЈА

Системске операције се могу састоји од више: а) **операција одржавања базе података** (убаци, избаци и промени) и/или б) **операција извештавања** (прикажи).
У току извршења системске операције над структуром софтверског система (односно над базом података) подаци (објекти у оперативној меморији и слогови у бази података) морају да остану конзистентни, односно морају да буду задовољена вредносна и структурна ограничења дефинисана над подацима.
Вредносна ограничења се односе на дозвољене вредности атрибута доменских класа (табела) и она се деле на:
а) проста вредносна ограничења - ограничења везана за домен (тип) атрибута и вредност атрибута.
б) сложена вредносна ограничења - ограничења везана за међузависност атрибута.

Структурна ограничења су дефинисана преко кардиналности пресликавања између доменских класа (табела).

При извршењу операција **убаци** и **промени** објекат (слог) проверавају се и вредносна и структурна ограничења. Ове провере се раде у одељку **предуслови** код уговора за системске операције. У одељку **постуслови** се наводи резултат операција убаци (нпр. *Направљен је нови рачун*), и промени (*нпр. Рачун је сторниран*, *Израчуната је укупна вредност рачуна*,...).

Убаци (insert)
Предуслови: *Вредносна и структурна ограничење морају бити задовољена.*
Постуслови: *Наводи се резултат операције.*

Промени (update)
Предуслови: *Вредносна и структурна ограничење морају бити задовољена.*
Постуслови: *Наводи се резултат операције.*

При извршењу операције **обриши** објекат (слог) проверавају се структурна ограничења. Ова провера се ради у одељку **предуслови** или у одељку **постуслови** код уговора за системске операције. У одељку **постуслови** се наводи резултат операција обриши (нпр. *Обрисан је предмет*).

Избаци (delete)
Предуслови: *Структурна ограничење морају бити задовољена.*
Постуслови: *Наводи се резултат операције.*

При извршењу операције **извештавања** не проверавају се вредносна и структурна ограничења. Код уговора за системске операције у одељцима **постуслови** и **предуслови** ништа се не наводи.

Прикажи (select)
Предуслови: /
Постуслови: /

2.2.4 СТРУКТУРА СОФТ. СИСТЕМА - КОНЦЕПТУАЛНИ (ДОМЕНСКИ) МОДЕЛ

Структура софтверског система се описује помоћу концептуалног модела. Наводимо дефиниције концептуалног модела и његових елемената.

Дефиниција АН7: Концептуални модел описује концептуалне класе домена проблема. Концептуални модел садржи концептуалне класе (доменске објекте) и асоцијације између концептуалних класа.Често се за концептуалне моделе каже да су то **доменски модели** или **модели објектне анализе.**

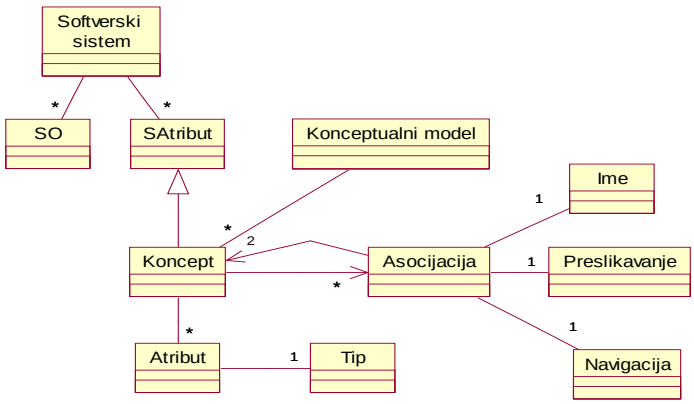
Дефиниција АН8: Концепти (концептуалне класе) представљају атрибуте[9] софтверског система. То значи да концепти описују структуру софтверског система. Концептуалне класе састоје се од атрибута, који описују особине класе. Концептуалне класе треба разликовати од софтверских класа.

Дефиниција АН9: Атрибути представљају особине која се придружују до концептуалних класа. Сваки од атрибута је везан за одређени тип података.
Атрибут има конкретну вредност за конкретно појављивање концептуалне класе.

Дефиниција АН10: Асоцијација је веза између концептуланих класа. Сваки крај асоцијације представља **улогу** концепта који учествује у асоцијацији. Улога садржи име, пресликавање и навигацију.
Име улоге је засновано на формату: ИмеКонцКласе1 – Глагол – ИмеКонцКласе2, где глагол описује однос између концептуалних класа у датом контексту.
Пресликавање дефинише колико много појављивања концептуалне класе А може бити придружено једном појављивању концептуалне класе Б.
Навигација указује на једносмерне везе између концептуалних класа.
Између концептуалних класа може постојати више асоцијација.

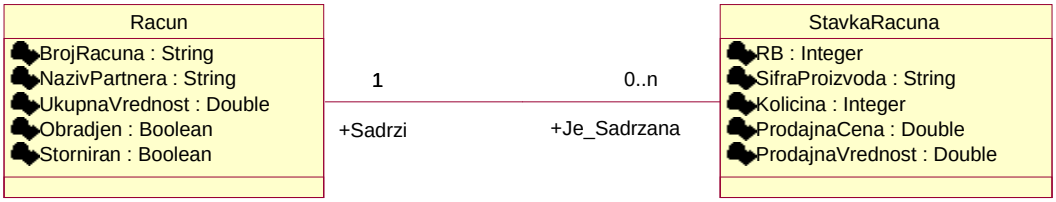
Објашњење дефиниција концептуалног модела и његових елемената

Софтверски систем се састоји од атрибута (САтрибут) и системских операција (СО). Концепти представљају реализацију атрибута софтверског система.

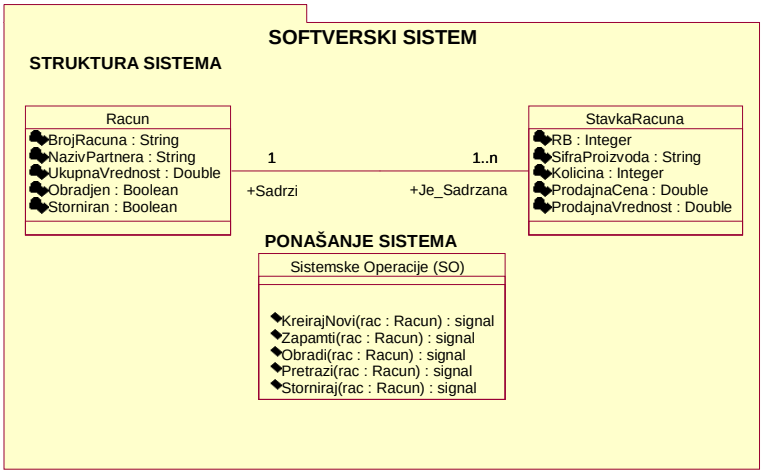


[9] Koncepti se mogu **uočiti** iz UA koji se prosleđuju do softverskog sistema. Međutim treba naglasiti da UA nisu koncepti. UA su izvan softverskog sistema i oni predstavljaju ulaz u softverski sistem. Koncepti su unutar softverskog sistema i oni predstavljaju strukturu softverskog sistema.

Конкретан пример концептуалног модела



Као резултат анализе сценарија СК и прављења концептуалног модела добија се **логичка структура и понашање софтверског система**:



2.2.5 СТРУКТУРА СОФТ. СИСТЕМА - РЕЛАЦИОНИ МОДЕЛ

На основу концептуалног модела може се направити релациони модел, који ће да представља основу за пројектовање релационе базе података[Ullman].

На основу датог концептуалног модела (Racun, StavkeRacuna) прави се **релациони модел**:

Racun(BrojRacuna, NazivPartnera,UkupnaVrednost,Obradjen,Storniran);

StavkaRacuna(BrojRacuna,RB,SifraProizvoda,Kolicina,ProdajnaCena,ProdajnaVrednost)

Табела Racun		**Просто вредносно ограничење**		**Сложено вредносно ограничење**		**Структурно ограничење**
Atributi	**Име**	**Тип атрибута**	**Вредност атрибута**	**Међузав. атрибута једне табеле**	**Међузав.атрибута више табела**	INSERT / UPDATE CASCADES StavkeRacuna DELETE CASCADES StavkeRacuna
	BrojRacuna	String	not null			
	NazivPartnera	String				
	UkupnaVrednost	Double	(default:0)		UkupnaVrednost= SUM (StavkaRacuna.ProdajnaVrednost)	
	Obradjen	Boolean	(default: false)			
	Storniran	Boolean	(default: false)			

Код сложених објеката (као што је Racun) треба узети следеће ограничење у обзир:
UPDATE Racun ----> DELETE StavkeRacuna (у бази) ----> INSERT StavkeRacuna (из оперативне меморије)

Табела StavkaRacuna		**Просто вредносно ограничење**		**Сложено вредносно ограничење**		**Структурно ограничење**
Atributi	**Име**	**Тип атрибута**	**Вредност атрибута**	**Међузав. атрибута једне табеле**	**Међузав. атрибута више табела**	INSERT RESTRICTED Racun UPDATE RESTRICTED Racun DELETE /
	BrojRacuna	String	Not null			
	RB	Integer	Not null and > 0			
	SifraProizvoda	String				
	Kolicina	Integer	>0 (default:0)			
	ProdajnaCena	Double	>0 (default:0)			
	ProdajnaVrednost	Double	(default:0)	ProdajnaVrednost = Kolicina*ProdajnaCena		

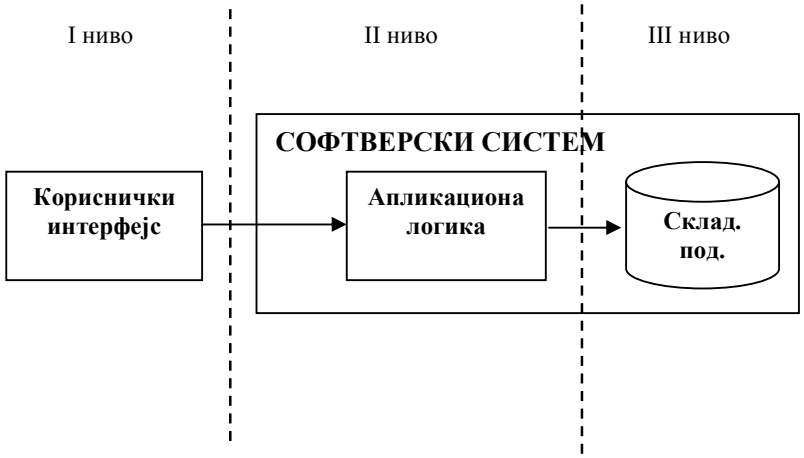
2.3 Пројектовања

Фаза пројектовања описује физичку структуру и понашање софтверског система (архитектуру софтверског система). Пројектовање архитектуре софтверског система обухвата пројектовање корисничког интерфејса, апликационе логике и складишта података. Пројектовање корисничког интерфејса обухвата пројектовање екранских форми и контролера корисничког интерфејса. У оквиру апликационе логике се пројектују контролер апликационе логике, пословна логика и брокер базе података. Пројектовање пословне логика обухвата пројектовање логичке структуре и понашања софтверског система.

АРХИТЕКТУРА СОФТВЕРСКОГ СИСТЕМА
Пре него што кренемо на пројектовање структуре и понашања софтверског система потребно је да се дефинише архитектура софтверског система. Ми ћемо користити класичну тронивојску архитектуру [TK78,Garther 95].

Дефиниције и правила архитектуре софтверског систа,

Деф. PRAR1: Тронивојска архитектура се састоји из следећих ниво:
1. **Корисничког интерфејса** који представља улазно – излазну репрезентацију софтверског система.
2. **Апликационе логике** која описује структуру и понашање софтверског система.
3. **Складишта података** који чува стање атрибута софтверског система.



Слика ТНА: Тронивојска архитектура

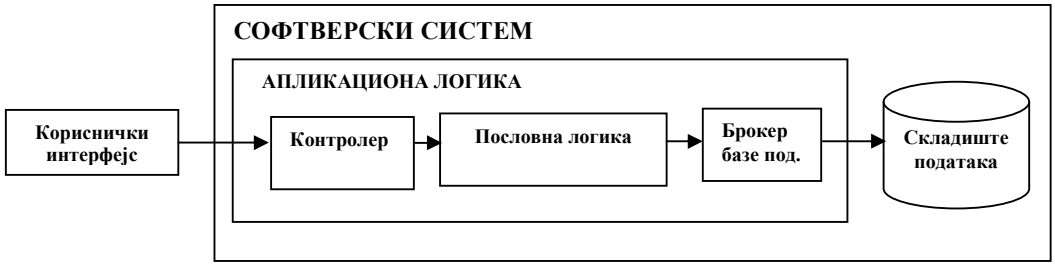
Правило PRpARH1: Апликациона логика се пројектује независно од корисничког интерфејса и обрнуто.

Правило PRpARH2: Апликациона логика може да има различите улазно-излазне репрезентације.

Правило PRpARH3 (Model-View Separation Principle): Апликациона логика (модел) нема знања о томе где се налази кориснички интерфејс (поглед).

Деф. ПРАР2: На основу **тронивојске архитектуре** су направљени савремени **апликациони сервери**.

Деф. ПРАР3: Апликациони сервери су одговорни да обезбеде сервисе који ће да омогуће реализацију апликационе логике софтверског система. Сваки апликациони сервер се састоји из три основна дела:
1. део за комуникацију са клијентим (контролер)
2. део за комуникацију са неким складиштем података (база података или датотечни систем)
3. део који садржи пословну логику



Деф. ПРАР4: Контролер је одговоран да прихвати захтев за извршење системске операције од клијента и да га проследи до пословне логике која је одговорна за извршење системске операције.

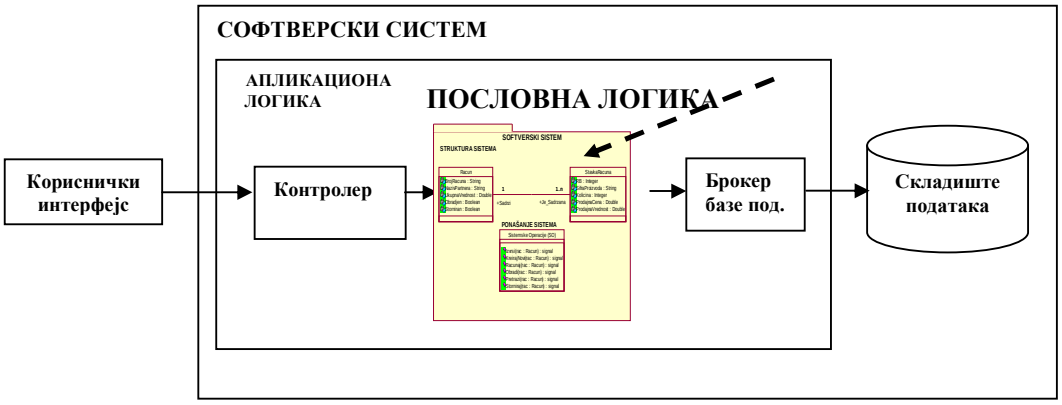
Деф. ПРАР5: Пословна логика је описана са структуром (доменским класама) и понашањем (системским операцијама).

Деф. ПРАР6: Брокер базе података је одговоран за комуникацију између пословне логике и складишта података.

У даљем тексту ћемо пројектовати сваки од наведених елемената тронивојске архитектуре:
- контролер
- пословна логика – доменске класе
- пословна логика – системске операције
- датабасе брокер
- складиште података
- кориснички интерфејс

Из наведеног можемо да закључимо да смо у фазама прикупљања захтева и анализе дали спецификацију структуре и понашања софтверског система, односно **спецификацију пословне логике софтверског система** (Slika ASSPL).

Из наведеног можемо да закључимо да смо у фазама прикупљања захтева и анализе дали спецификацију структуре и понашања софтверског система, односно **спецификацију пословне логике софтверског система**.



Слика **АССПЛ**: Архитектура софт. система – Пословна логика

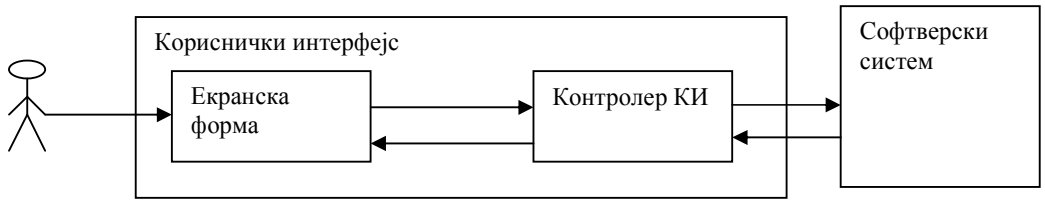
2.3.1 Пројектовање корисничког интерфејса

Кориснички интерфејс, сходно Деф. Лаз1, представља реализацију улаза и/или излаза софтверског система. Пре пројектовања корисничког интерфејса објаснићемо делове корисничког интерфејса (његову структуру), начин њихове међусобне комуникације и начин комуникације корисничког интерфејса са софтверским системом (Слика СКИ).

Кориснички интерфејс се састоји од:

1. Екранске форме која је одговорна да:
   а) прихвата податке које уноси актор,
   б) прихвата догађаје које прави актор,
   ц) позива контролера графичког интерфејса, прослеђујући му прихваћене податке
   д) приказује податке које је добио од контролера графичког интерфејса.

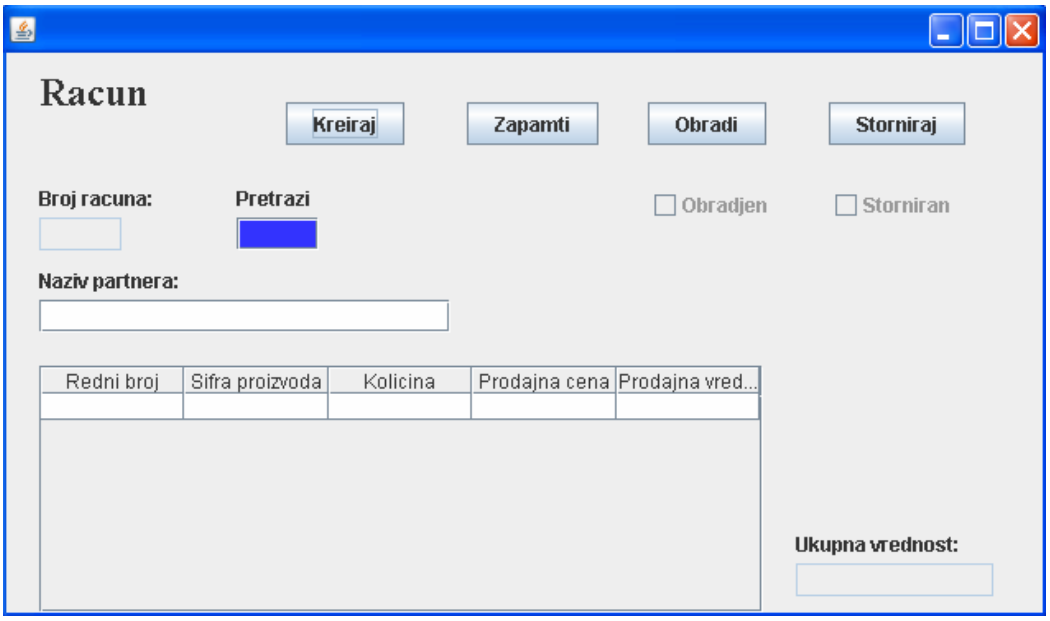
2. Контролера корисничког интерфејса који је одговоран да:
   а) прихвати податке које шаље екранска форма,
   б) конвертује податке (који се налазе у графичким елементима) у објекат који представља улазни аргумент СО која ће бити позвана,
   ц) шаље захтев за извршење СО до апликационог сервера (софтверског система),
   д) прихвата објекат (излаз) софтверског система који настаје као резултат извршења СО и
   е) конвертује објекат у податке графичких елемената.



Слика **СКИ:** Структура корисничког интерфејса

2.3.1.1 ПРОЈЕКТОВАЊЕ ЕКРАНСКЕ ФОРМЕ

Екранска форма треба, за наведени пример, да има следећи изглед:



Кориснички интерфејс је дефинисан преко скупа екранских форми. Сценарија коришћења екранских форми је директно повезан са сценаријима случајева коришћења.
Постоје два аспекта пројектовања екранске форме:

a) Пројектовање сценарија СК који се изводе преко екранске форме.
b) Пројектовање метода екранске форме.

Пројектовање сценарија СК

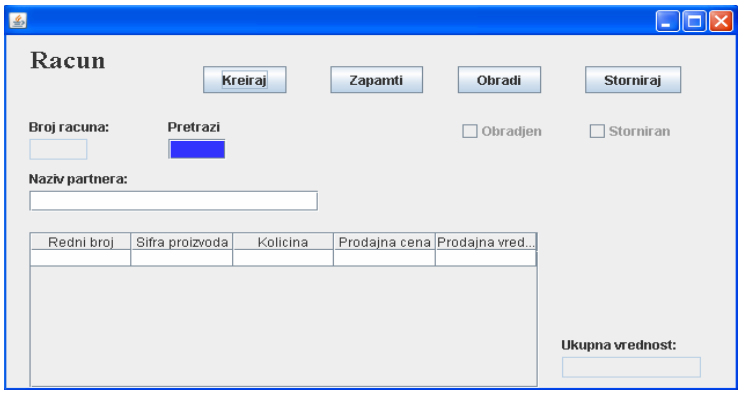
СК1: Случај коришћења – Креирање новог рачуна

Назив СК
Креирање новог рачуна

Актори СК
Продавац

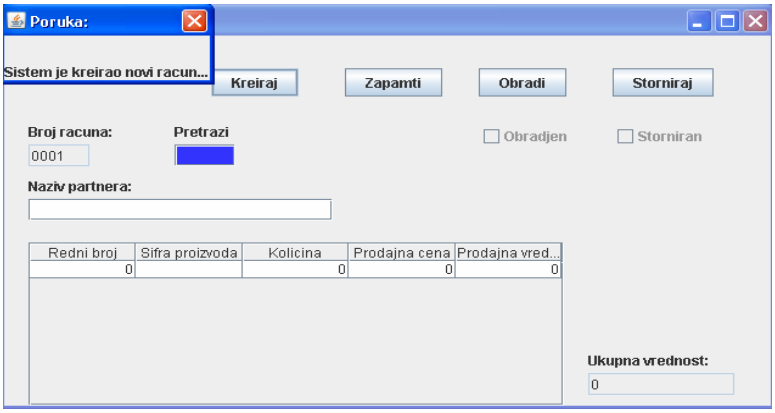
Учесници СК
Продавац и систем (програм)

Предуслов: Систем је укључен и продавац је улогован под својом шифром. Систем приказује форму за рад са рачуном.

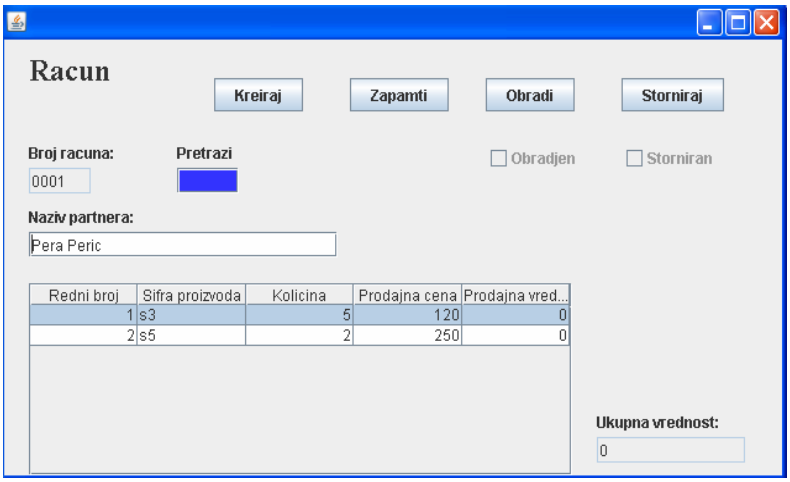


Основни сценарио СК

1. Продавац **позива** систем да креира нови рачун. (АПСО)
*Опис акције: Продавац кликом на дугме "Kreiraj" позива системску операцију **kreirajNovi (Racun)** која прави нови рачун.*
2. Систем **креира** нови рачун. (СО)
3. Систем **приказује** продавцу нови рачун и поруку: "Систем је креирао нови рачун". (ИА)



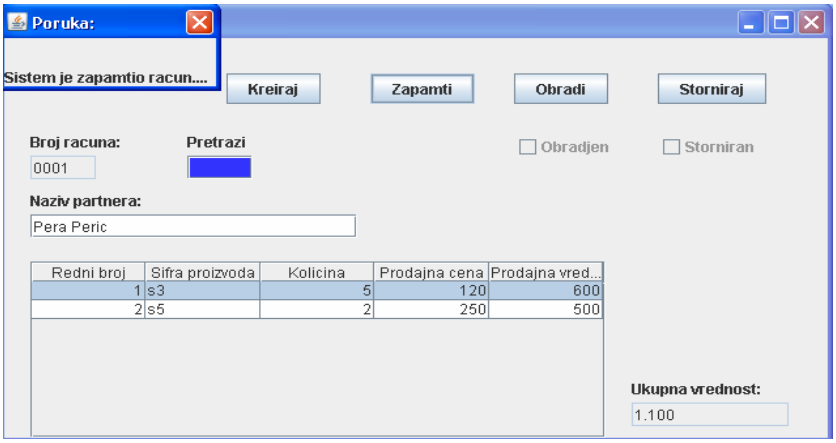
4. Продавац **уноси** податке у нови рачун. (АПУСО)



5. Продавац **контролише** да ли је коректно унео податке у нови рачун. (АНСО)
6. Продавац **позива** систем да запамти податке о рачуну. (АПСО)

*<u>Опис акције:</u> Продавац кликом на дугме "Zapamti " позива системску операцију **Zapamti (Racun)** која памти нови рачун.*

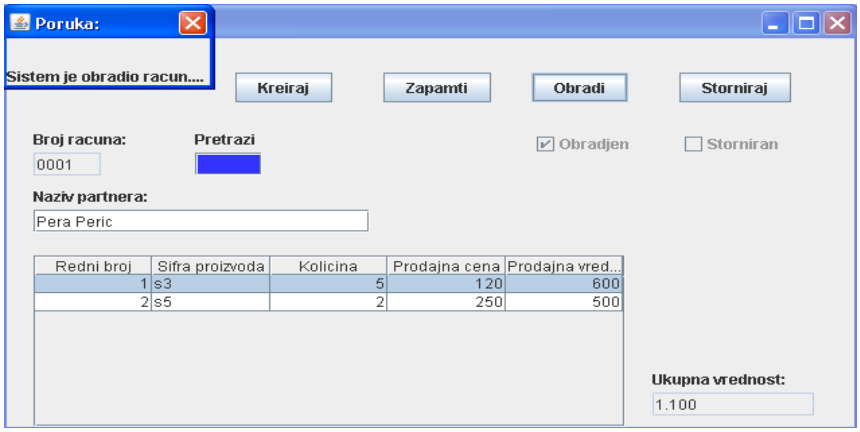
7. Систем **памти** податке о рачуну. (СО)
8. Систем **приказује** продавцу запамћени рачун и поруку: "Систем је запамтио рачун". (ИА)



9. Продавац **позива** систем да обради рачун. (АПСО)

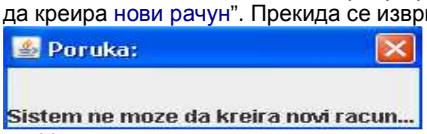
*<u>Опис акције:</u> Продавац кликом на дугме "Obradi " позива системску операцију **Obradi (Racun)** која обрађује нови рачун.*

10. Систем **обрађује** рачун.(СО)
11. Систем **приказује** продавцу обрађен рачун и поруку: "Систем је обрадио рачун".(ИА)

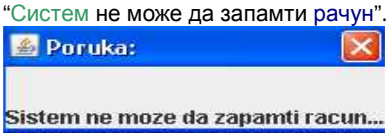


Алтернативна сценарија

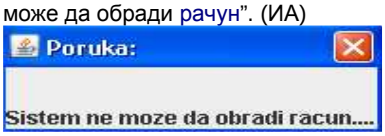
3.1 Уколико систем не може да креира рачун он приказује продавцу поруку: "Систем не може да креира нови рачун". Прекида се извршење сценариа. (ИА)



8.1 Уколико систем не може да запамти податке о рачуну он приказује продавцу поруку "Систем не може да запамти рачун". Прекида се извршење сценариа. (ИА)



11.1 Уколико систем не може да обради рачун он приказује продавцу поруку: "Систем не може да обради рачун". (ИА)



На сличан начин за наведени пример унећемо још 2 рачуна.

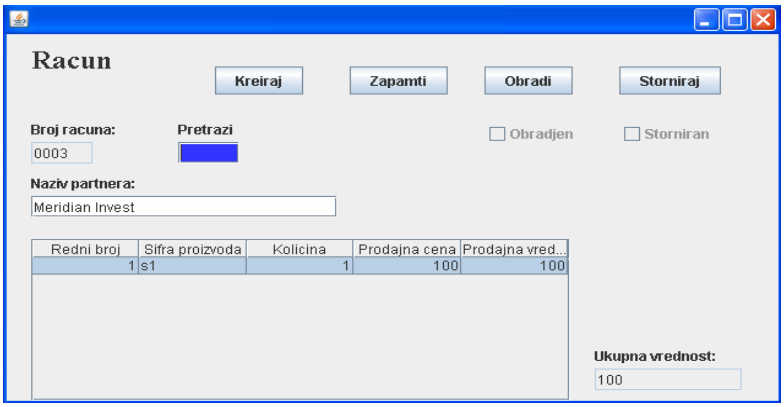
СК2: Случај коришћења – Претраживање рачуна

Назив СК
Претраживање рачуна

Актори СК
Корисник

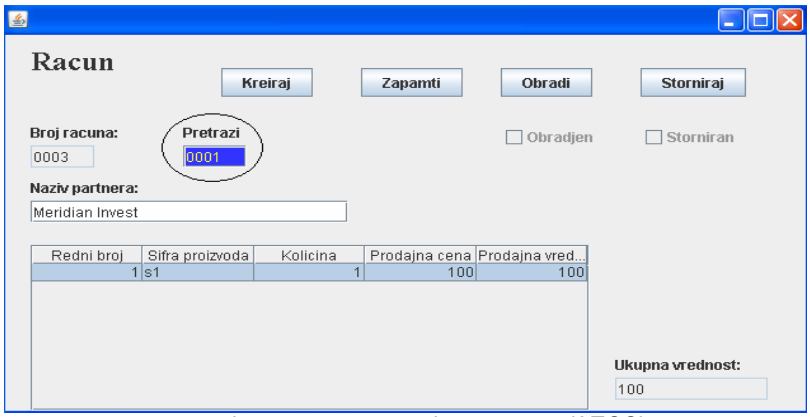
Учесници СК
Корисник и систем (програм)

Предуслов: Систем је укључен и корисник је улогован под својом шифром. Систем приказује форму за рад са рачуном.



Основни сценарио СК

1. Корисник **уноси** вредност по којој претражује рачун. (АПУСО)
Опис акције: Корисник уноси вредност у поље под називом Pretrazi.

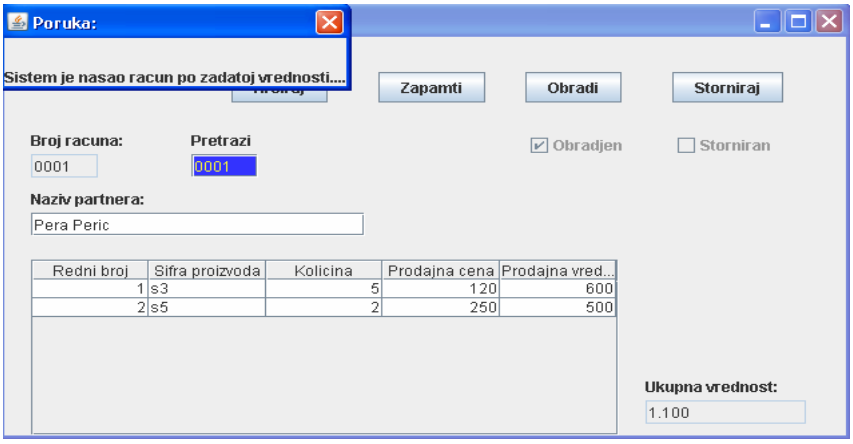


2. Корисник **позива** систем да нађе рачун по задатој вредности. (АПСО)

*Опис акције: Корисник након уноса вредности у поље под називом Pretrazi притиска типку <Enter> и позива системску операцију **Pretrazi (Racun)**.*

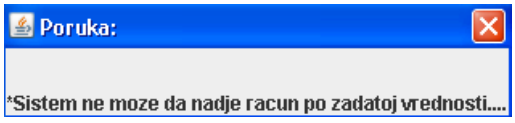
3. Систем **тражи** рачун по задатој вредности. (СО)

4. Систем приказује кориснику податке о рачуну и поруку: "Систем је нашао рачун по задатој вредности". (ИА)



Алтернативна сценарија

4.1 Уколико систем не може да нађе рачун он приказује кориснику поруку: "Систем не може да нађе рачун по задатој вредности". (ИА)



СК3: Случај коришћења – Сторнирање рачуна

Назив СК
Сторнирање рачуна

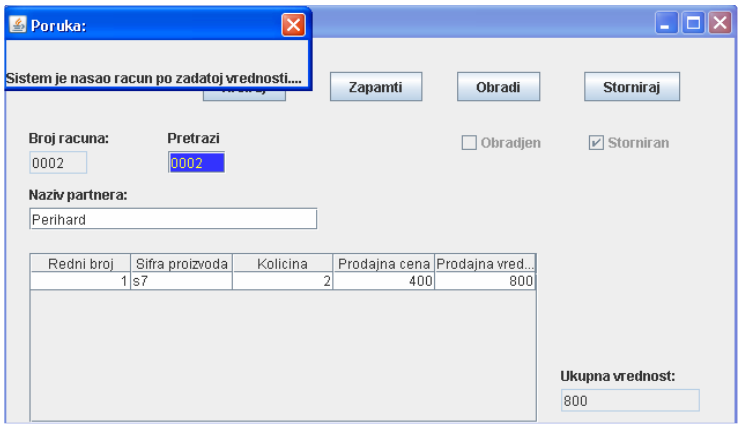
Актори СК
Корисник

Учесници СК
Корисник и систем (програм)

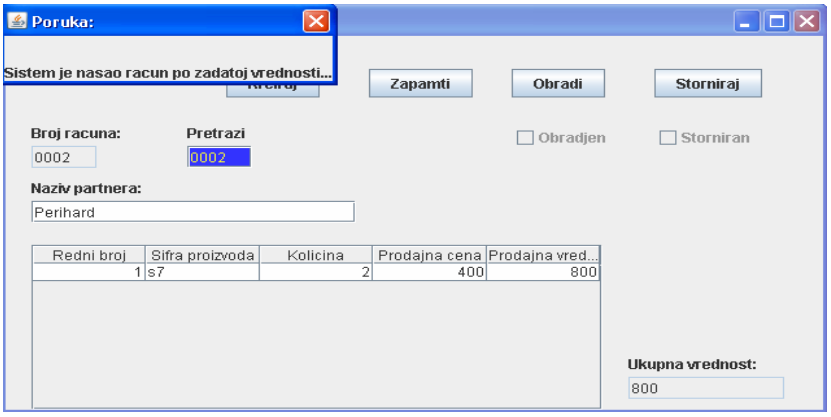
Предуслов: Систем је укључен и корисник је улогован под својом шифром. Систем приказује форму за рад са рачуном.

Основни сценарио СК

1. Корисник **уноси** вредност по којој претражује рачун. (АПУСО)
   Опис акције: Корисник уноси вредност у поље под називом Pretrazi.

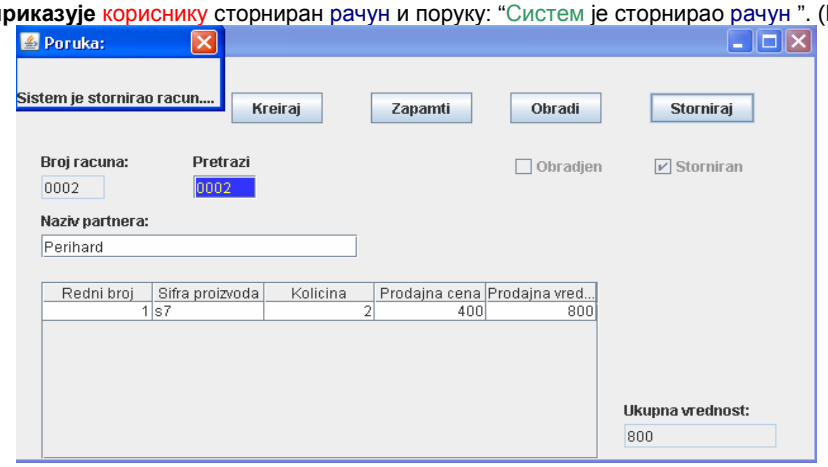


2. Корисник **позива** систем да нађе рачун по задатој вредности. (АПСО)
   *Опис акције: Корисник након уноса вредности у поље под називом Pretrazi притиска типку <Enter> и позива системску операцију **Pretrazi (Racun)**.*
3. Систем **тражи** рачун по задатој вредности. (СО)
4. Систем приказује кориснику рачун и поруку: "Систем је нашао рачун по задатој вредности". (ИА)



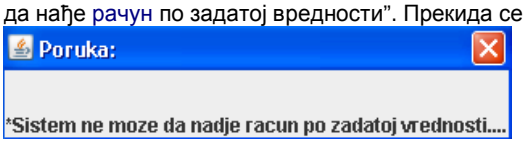
5. Корисник **позива** систем да сторнира задати рачун. (АПСО)
   *Опис акције: Корисник кликом на дугме "Storniraj" позива системску операцију **Storniraj (Racun)**.*
6. Систем **сторнира** рачун. (СО)

7. Систем **приказује** кориснику сторниран рачун и поруку: "Систем је сторнирао рачун ". (ИА)

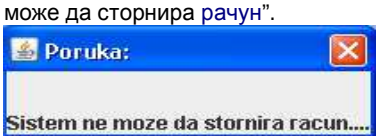


Алтернативна сценарија

4.1 Уколико систем не може да нађе рачун он приказује кориснику поруку: "Систем не може да нађе рачун по задатој вредности". Прекида се извршење сценариа. (ИА)



7.1 Уколико систем не може да сторнира рачун он приказује кориснику поруку: "Систем не може да сторнира рачун".



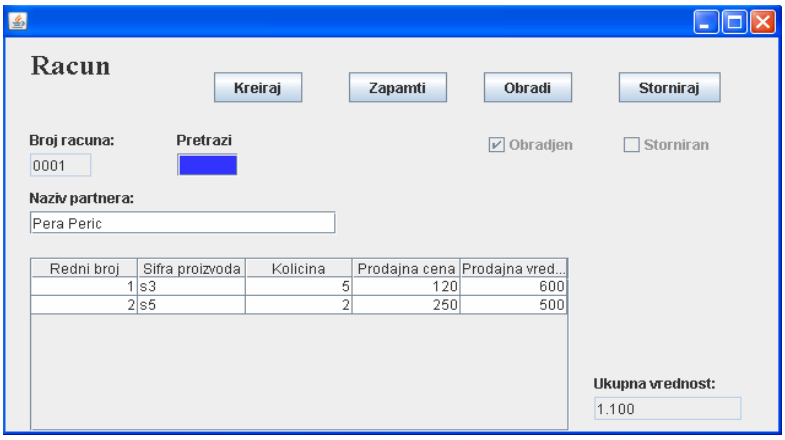
СК4: Случај коришћења – Промена рачуна

Назив СК
Промена рачуна

Актори СК
Продавац

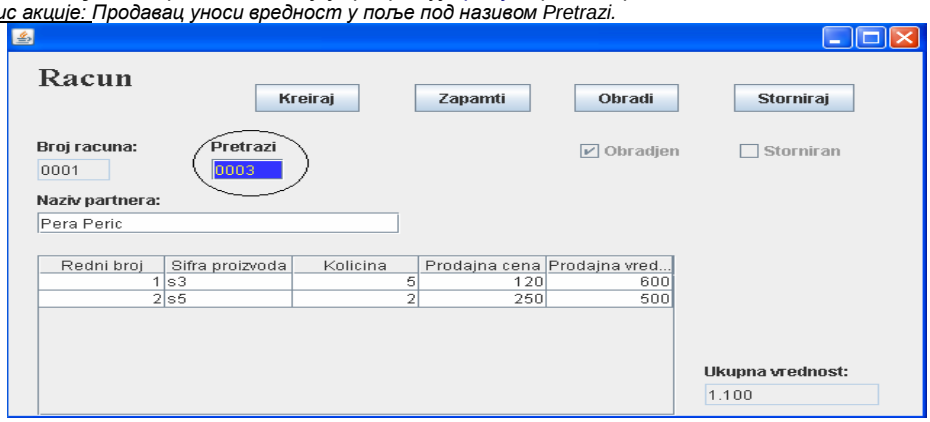
Учесници СК
Продавац и систем (програм)

Предуслов: Систем је укључен и продавац је улогован под својом шифром. Систем приказује форму за рад са рачуном.

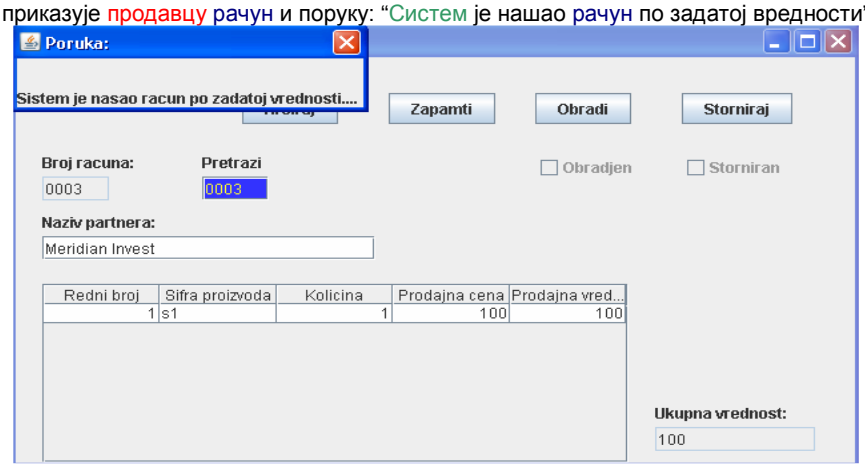


Основни сценарио СК

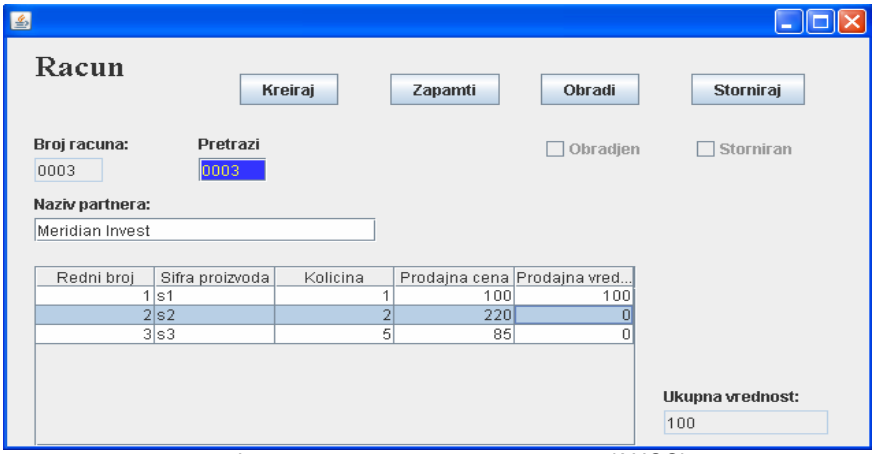
1. Продавац **уноси** вредност по којој претражује рачун. (АПУСО)
   Опис акције: Продавац уноси вредност у поље под називом Pretrazi.



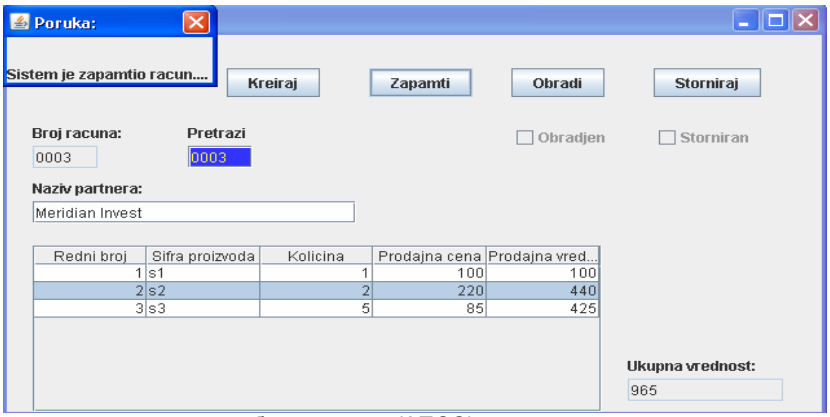
2. Продавац **позива** систем да нађе рачун по задатој вредности. (АПСО)
3. Систем **тражи** рачун по задатој вредности. (СО)
4. Систем приказује продавцу рачун и поруку: "Систем је нашао рачун по задатој вредности". (ИА)



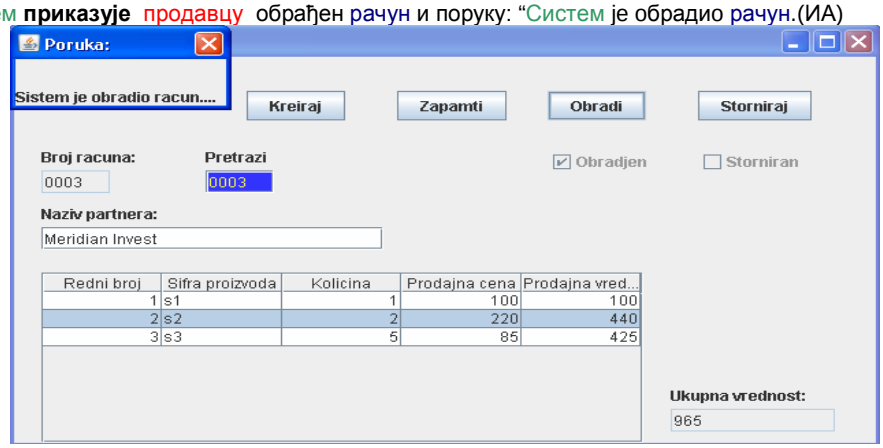
5. Продавац **уноси (мења)** податке о рачуну. (АПУСО)



6. Продавац **контролише** да ли је коректно унео податке о рачуну. (АНСО)
7. Продавац **позива** систем да запамти податке о рачуну. (АПСО)
   *Опис акције: Продавац кликом на дугме "Zapamti " позива системску операцију **Zapamti (Racun)**.*
8. Систем **памти** податке о рачуну. (СО)
9. Систем **приказује** продавцу запамћени рачун и поруку: "Систем је запамтио рачун." (ИА)

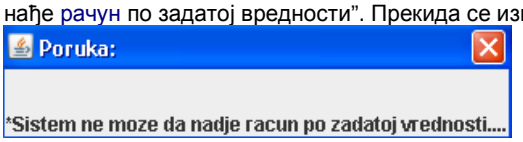


10. Продавац **позива** систем да обради рачун. (АПСО)
    Опис акције: *Продавац кликом на дугме "Obradi " позива системску операцију* ***Obradi (Racun)*** *која обрађује нови рачун.*
11. Систем **обрађује** рачун.(СО)
12. Систем **приказује** продавцу обрађен рачун и поруку: "Систем је обрадио рачун.(ИА)

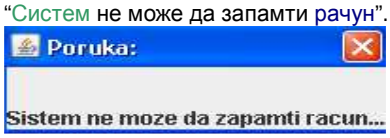


Алтернативна сценарија

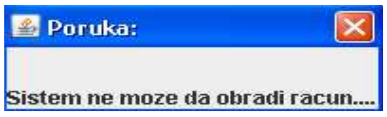
4.1 Уколико систем не може да нађе рачун он приказује продавцу поруку: "Систем не може да нађе рачун по задатој вредности". Прекида се извршење сценариа. (ИА)



9.1 Уколико систем не може да запамти податке о рачуну он приказује продавцу поруку "Систем не може да запамти рачун". Прекида се извршење сценариа. (ИА)



12.1 Уколико систем не може да обради рачун он приказује продавцу поруку: "Систем не може да обради рачун". (ИА)



Пројектовање метода екранске форме

Постоје две класе које се користе у пројектовању метода екранске форме: **OpstaEkranskaForma** и **EkranskaFormaRacun.** Абстрактна класа OpstaEkranskaForma садржи опште методе које су независне од екранске форме која ће да репрезентује неку доменску класу. Док класа EkranskaFormaRacun, садржи методе које приказују екранску форму која репрезентује конкретну доменску класу (у нашем примеру рачун) са свим припадајућим графичким елементима. Класа EkranskaFormaRacun је повезана са класом KontrolerKIRacun којој прослеђује графички објекат (при иницијализацији) и захтев за извршење системске операције.

```
abstract class OpstaEkranskaForma extends … // (navodi se ime klase koja omogućava kreiranje
                                              //   ekranske forme)
{
   … // navode se opšte metode ekranske forme.
  abstract OpstiDomenskiObjekat kreirajObjekat();
}


class EkranskaFormaRacun extends OpstaEkranskaForma
 { KontrolerKIRacun kkir;


   // Glavni program
   public static void main(String args[])
     { EkranskaFormaRacun EF = new EkranskaFormaRacun();
     }

   // 1. Konstruktor ekranske forme
   public EkranskaFormaRacun ()
     {   kreirajKomponenteEkranskeForme();          // 1.1
         pokreniMenadzeraRasporedaKomponeti();   // 1.2
         postaviImeForme();                          // 1.3
         postaviTextFieldBrojRacuna();             // 1.4
         postaviTextFieldNazivPartnera();          // 1.5
         postaviTextFieldUkupnaVrednost();       // 1.6
        postaviTextFieldPretrazivanje();           // 1.7
        postaviCheckBoxObradjen();                 // 1.8
        postaviCheckBoxStorniran();                // 1.9
        postaviDugmeKreiraj();                     // 1.10
       postaviDugmeObradi();                       // 1.11
       postaviDugmeStorniraj();                    // 1.12
       postaviDugmeZapamti();                      // 1.13
       postaviTabelu();                            // 1.14
       inicijalizacijaKontrolera();                // 1.15
    }


…
void postaviDugmeKreiraj()
 {   …
     // Kada se klikne na dugne poziva se:
    String signal = kkir.SOKreirajNovi();
     …
}


void postaviDugmeObradi()
 {  …
    // Kada se klikne na dugne poziva se:
    String signal = kkir.SOObradi();
    …
}
```

```
void  postaviDugmeStorniraj()
{ ...
   // Kada se klikne na dugne poziva se:
   String signal = kkir.SOStorniraj();
  …
 }


void postaviDugmeZapamti()
 { …
   // Kada se klikne na dugne poziva se:
   String signal = kkir.SOZapamti();
   …
}


void postaviTabelu()
{  ...
   // Kada se pritisne neka od tipki na tabeli poziva se:
         String signal = kkir.pritisakTipke(evt);
  …
 }


// 1.15 Inicijalizacija KontroleraKI
// Pri inicijalizaciji, kontroler dobija referencu na graficki objekat (this).
void inicijalizacijaKontrolera()
      {  kkir = new KontrolerKIRacun (this); }

OpstiDomenskiObjekat kreirajObjekat() {return new Racun();}

}

}
```

2.3.1.2 ПРОЈЕКТОВАЊЕ КОНТРОЛЕРА КОРИСНИЧКОГ ИНТЕРФЕЈСА

Контролер корисничког интерфејса треба пројектовати тако да има општи део (***OpstiKontrolerKI***) који је независтан од екранске форме преко које се извршава сценарију случаја коришћења и конректни део који је везан за домен екранске форме (**KontrolerKIRacun**).

Општи контролер:

a) успоставља везу између екранске форме и апликационе логике.
b) прихвата од екранске форме захтев за извршење системске операције.
c) креира доменски објекат.
d) прослеђује захтев за извршење системске операције и доменске објекте до апликационог сервера (апликационе логике).
e) прихвата доменске објекте и сигнале (о успешности извршења СО) које је вратио апликациони сервер као резултат извршења системске операције.

Конкретни контролер:

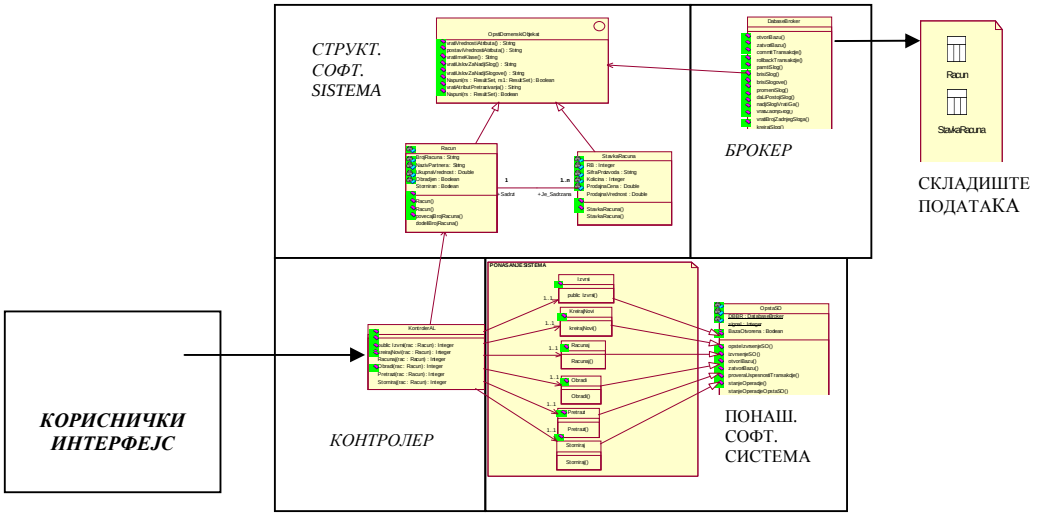
a) прихвата од екранске форме графичке објекте.
b) конвертује графичке објекте у доменске објекте који ће бити прослеђени преко мреже до апликационог сервера.
c) конвертује доменске објекте у графичке објекте и прослеђује их до екранске форме.

```
abstract class OpstiKontrolerKI
 { AplikacionaLogika al;
   String signal;
   OpstiDomenskiObjekat odo;
   OpstaEkranskaForma oef;

// a) успоставља везу између екранске форме и апликационе логике преко сокета.
   OpstiKontrolerKI()
     { UspostaviVezuIzmeđuEkranskeFormeIAplikacioneLogike();  }

   public String pritisakTipke(KeyEvent evt)
    {  OdredjujeSeNacinObradiTipkiPriRaduSaTabelom();  }

// b) прихвата од екранске форме захтев за извршење системске операције.
   public String SOPretrazi()
   {  // c) креира доменски објекат.
      odo = oef.kreirajObjekat();
      KonvertujGrafickiObjekatUDomenskiObjekat();
      signal = pozivSO("Pretrazi");
      KonvertujDomenskiObjekatUGrafickiObjekat();
      return signal;
  }


  public String SOKreirajNovi()
   {  odo = oef.kreirajObjekat();
      signal = pozivSO("kreirajNovi");
      KonvertujObjekatUGrafickeKomponente();
      return signal;
}

  public String SOZapamti()
   {  odo = oef.kreirajObjekat();
      KonvertujGrafickiObjekatUDomenskiObjekat ();
      signal = pozivSO("Zapamti");
      KonvertujDomenskiObjekatUGrafickiObjekat();
      return signal;
   }

  public String SOStorniraj()
   {  odo = oef.kreirajObjekat();
      KonvertujGrafickiObjekatUDomenskiObjekat();
      signal = pozivSO("Storniraj");
      KonvertujDomenskiObjekatUGrafickiObjekat();
```

```
      return signal;
   }

   public String SOObradi()
    {   odo = oef.kreirajObjekat();
        KonvertujGrafickiObjekatUDomenskiObjekat();
        signal = pozivSO("Obradi");
        KonvertujDomenskiObjekatUGrafickiObjekat();
        return signal;
     }

// d) прослеђује захтев за извршење системске операције и доменске објекте до апликационог
// сервера (апликационе логике).
// e) прихвата доменске објекте и сигнале (о успешности извршења СО) које је вратио
// апликациони сервер као резултат извршења системске операције.
String pozivSO(String nazivSO)
{  signal = PozivAplikacionogServera(odo);
   return signal;
}

abstract public void KonvertujGrafickiObjekatUDomenskiObjekat();
abstract public void KonvertujDomenskiObjekatUGrafickiObjekat();
}



class KontrolerKIRacun extends OpstiKontrolerKI
{

// a) прихвата од екранске форме графичке објекте.
  KontrolerKIRacun(EkranskaFormaRacun efr)  {oef = efr;}

// b) конвертује графичке објекте  у доменске објекте који ће бити прослеђени преко мреже до
// апликационог сервера.
public void KonvertujGrafickiObjekatUDomenskiObjekat()
   {
      Racun rac = (Racun) odo;
      EkranskaFormaRacun efr = (EkranskaFormaRacun) oef;
      KonvertujeElementeGrafickogObjektauAtributeDomenskogObjekta();
      KonvertujeRedoveTabeleStavkeRacunaUNizObjekataStavkeRacuna();
   }


// c) конвертује доменске објекте у графичке објекте и прослеђује их до екранске форме.
 public void KonvertujDomenskiObjekatUGrafickiObjekat()
   {  Racun rac = (Racun) odo;
      EkranskaFormaRacun efr = (EkranskaFormaRacun) oef;
      KonvertujeAtributeDomenskogObjektaUElementeGrafickogObjekta();
      KonvertujeNizObjekataStavkeRacunaURedoveTabeleStavkeRacuna();
   }
}
```

Након пројектовања корисничког интерфејса добија се следећи дијаграм класа:

Кориснички интерфејс у контексту архитектуре софтверског система може се представити на следећи начин:



2.3.2 Пројектовање апликационе логике

2.3.2.1 Контролер апликационе логике

Контролер апликационе логике треба да подигне серверски сокет који ће да ослушкује мрежу. Када клијент (клијентски сокет) успостави конекцију са контролером (серверским сокетом), тада контролер треба да генерише нит која ће успоставити двосмерну везу са клијентом (улазну и излазну). Слање и примање података од клијента се остварује преко сокета.
Клијент шаље захтев за извршење неке од СО до одговарајуће нити (коју смо назвали "нит клијента"), која је повезана са тим клијентом. "Нит клијента" прима захтев и даље га преусмерава до класа које су одговорне за извршење СО. Након извршења СО резултат се враћа до апликационе логике, односно до "нити клијента", која тај резултат шаље назад до клијента.

Дајемо приказ пројектоване класа *KontrolerAL* и *NitKlijenta.*

```
class KontrolerPL // Kontroler poslovne logike
{
   void main(String[] args)
            {
            serverskiSoket = podizanjeServerskogSoketa();
            for(…)
              {  klijentskiSoket = serverskiSoket.osluskivanjeMreze();
                 NitKlijenta.kreiranjeNitiKlijenta(klijentskiSoket);

              }

            }


class NitKlijenta {

void kreiranjeNitiKlijenta (klijentskiSoket)
  {
    povezivanjeNitiSaKlijentskimSoketom();
    Ulaz ul = kreiranjeUlazaPrekoKlijentskogSoketa();
    Izlaz iz = kreiranjeIzlazaPrekoKlijentskogSoketa();

    ImeOperacije = ul.PrihvatiImeOperacijeOdKlijenta();
    OpstiDomenskiObjekat odo = ul.PrihvatiDomenskiObjekatOdKlijenta();

    if  (ImeOperacije = "kreirajNovi")   signal = KreirajNovi.kreirajNovi(odo);
    if  (ImeOperacije = "Pretrazi")      signal = Pretrazi.Pretrazi(odo);
    if  (ImeOperacije = "Zapamti")       signal = Racunaj.Zapamti(odo);
    if  (ImeOperacije = "Obradi")        signal = Obradi.Obradi(odo);
    if  (ImeOperacije = "Storniraj")     signal = Storniraj.Storniraj(odo);

     izl.SlanjeDomenskogObjektaDoKlijenta(rac);
     izl.SlanjeSignalaDoKlijenta(signal);
  }
```

2.3.2.2 Пословна логика

2.3.2.2.1 Пројектовање понашања софтверског система – системске операције

Препорука ПР1: У почетку пројектовања СО треба треба направити концептуалне реализације (решења) за сваку СО.Концептуалне реализације требају да буду директно повезане са логиком проблема.

Препорука ПР2: Може се предпоставити да се подаци (стање софтверског система) чувају у бази. У том смислу могу се позвати неке од основних операција базе (insert,update,delete,find,...), без улажења у начин њихове реализације.

Препорука ПР3: Аспекти реализације који се односе на конекцију са базом, перзистентност и трансакције треба избећи у почетку пројектовања СО. У каснијим фазама наведени аспекти требају ортогонално да се повежу са пројектованим решењима СО, како би се логика решења проблема независно од њих развијала.

Препорука ПР4: Концептуалне реализација се могу описати преко објектног псеудокода, дијаграма сарадње [Larman,JPRS], секвенцних дијаграма[10] [Larman,JPRS], дијаграма активности, дијаграма прелаза стања или дијаграма структура [Budgen].

За сваки од уговора пројектује се концептуално решење.

Projektovanje konceptualnih rešenja SO

1. Уговор UG1: *KreirajNovi*

Операција: KreirajNovi (*OpstiDomenskiObjekat*):signal;
Веза са СК: СКП31
Предуслови:
Постуслови: *Направљен је нови доменски објекат.*.

```
class KreirajNovi extends OpstaSO
{
  public static String kreirajNovi(OpstiDomenskiObjekat odo)
    { KreirajNovi kn = new KreirajNovi();
      OpstaSO.transakcija = true;
      return OpstaSO.opsteIzvrsenjeSO(odo,kn);
    }

  // Prekrivanje metode klase OpstaSO
  boolean izvrsenjeSO(OpstiDomenskiObjekat odo)
   {  if (!BBP.kreirajSlog(odo))
        { BBP.dodajPorukuMetode("Sistem ne moze da kreira " + odo.vratiNazivNovogObjekta() + ".");
              return false;
        }
      BBP.dodajPorukuMetode("Sistem je kreirao " + odo.vratiNazivNovogObjekta() + ".");
      return true;
   }

}
```

[10] I sekvencni i dijagram saradnje čuvaju istu semantiku interakcije između objekata. Razlika između njih se ogleda u načinu njihovog predstavljanja.
Dijagrami saradnje opisuju interakciju između objekata u grafičkom ili mrežnom formatu, u kome objekti mogu da se postave bilo gde na dijagramu.
Sekvencni dijagram opisuje interakciju u fence(ograda) formatu, u kome se svaki sledeći objekat dodaje desno u odnosu na predhodni objekat.
Navedeni dijagrami imaju određene prednosti i nedostatke:

- **Sekvencni dijagram:**
  - prednosti: jasno se prikazuje sekvenca poruka u vremenu, jednostavno opisivanje.
  - nedostatak: kada se dodaje novi objekat zauzima se prostor u desno.
- **Dijagram saradnje:**
  - prednost: jednostavnije se dodaju novi objekti na dijagramu. Bolje se ilustruju kompleksne grane, ciklusi i konkurentno ponašanje.
  - nedostatak: Teškoće kod posmatranja sekvence akcija. Složenija notacija.

2. Уговор UG2: *Zapamti*

Операција: Racunaj(*OpstiDomenskiObjekat*):signal;
Веза са СК: *СК1, СК4*
Предуслови: *Ако је доменски објекат обрађен или сторниран не може се извршити системска операција. Просто вредносно ограничење над доменским објектом мора бити задовољено.*
Постуслови: *Запамћен је доменски објекат.*

```
class Zapamti extends OpstaSO
{
  public static String Zapamti(OpstiDomenskiObjekat odo)
   { Zapamti r = new Zapamti();
     OpstaSO.transakcija = true;
     return OpstaSO.opsteIzvrsenjeSO(odo,r);
   }

  // Prekrivanje metode klase OpstaSO
  boolean izvrsenjeSO(OpstiDomenskiObjekat odo)
   { if (!Preduslov(odo))
              { BBP.dodajPorukuMetode("Sistem ne moze da zapamti " + odo.vratiNazivObjekta() + "." +
                                                  odo.vratiImeKlase() + " je vec obradjen ili storniran.");
                return false;
              }

     if (!odo.vrednosnaOgranicenja())
              { BBP.dodajPorukuMetode("Sistem ne moze da zapamti " + odo.vratiNazivObjekta() + ".
                                                  Naruseno je vrednosno ogranicenje.");
                return false;
              }

      if (!BBP.brisiSlog(odo))
              { BBP.dodajPorukuMetode("Sistem ne moze da zapamti " + odo.vratiNazivObjekta() + ".");
                return false;
              }

       if (!BBP.pamtiSlozeniSlog(odo))
              { BBP.dodajPorukuMetode("Sistem ne moze da zapamti " + odo.vratiNazivObjekta() + ".");
                return false;
              }

      BBP.dodajPorukuMetode("Sistem je zapamtio " + odo.vratiNazivObjekta() + ".");
      return true;
    }

  private boolean Preduslov(OpstiDomenskiObjekat odo)
    { if ((BBP.vratiLogickuVrednostAtributa(odo,"Obradjen") == true) ||
          (BBP.vratiLogickuVrednostAtributa(odo,"Storniran") == true))
            { return false;
            }
      return true;
    }
}
```

3. Уговор UG3: *Obradi*

Операција: Obradi(*OpstiDomenskiObjekat*):signal;
Веза са СК: *СК1, СК4*
Предуслови: *Ако је доменски објекат обрађен или сторниран не може се извршити системска операција. Просто вредносно ограничење над доменским објектом мора бити задовољено.*
Постуслови: *Доменски објекат је обрађен.*

```
class Obradi extends OpstaSO
 {
  public static String Obradi(OpstiDomenskiObjekat odo)
   { Obradi ob = new Obradi();
     OpstaSO.transakcija = true;
     return OpstaSO.opsteIzvrsenjeSO(odo,ob);
   }


  // Prekrivanje metode klase OpstaSO
  boolean izvrsenjeSO(OpstiDomenskiObjekat odo)
   { if (!Preduslov(odo))
            { BBP.dodajPorukuMetode("Sistem ne moze da obradi " + odo.vratiNazivObjekta() + ".
              Dokument je vec obradjen ili storniran.");
              return false;
            }

     if (!odo.vrednosnaOgranicenja())
            { BBP.dodajPorukuMetode("Sistem ne moze da obradi " + odo.vratiNazivObjekta() + ".
                                     Naruseno je vrednosno ogranicenje.");
              return false;
            }

     odo.Obradi();

      if (!BBP.brisiSlog(odo))
            { BBP.dodajPorukuMetode("Sistem ne moze da obradi " + odo.vratiNazivObjekta() + ".");
              return false;
            }

      if (!BBP.pamtiSlozeniSlog(odo))
            { BBP.dodajPorukuMetode("Sistem ne moze da obradi " + odo.vratiNazivObjekta() + ".");
              return false;
            }

     BBP.dodajPorukuMetode("Sistem je obradio " + odo.vratiNazivObjekta() + ".");
     return true;
 }


  private boolean Preduslov(OpstiDomenskiObjekat odo)
    { if ((BBP.vratiLogickuVrednostAtributa(odo,"Obradjen") == true) ||
          (BBP.vratiLogickuVrednostAtributa(odo,"Storniran") == true))
           { return false;
           }
     return true;
     }
}
```

4. Уговор UG4: *Pretraži*

Операција: Pretraži (*OpstiDomenskiObjekat*):signal;
Веза са СК: *СК2, СК3, СК4*
Предуслови:
Постуслови:

```
class Pretrazi extends OpstaSO
 {
   public static String Pretrazi(OpstiDomenskiObjekat odo)
    { Pretrazi p = new Pretrazi();
              OpstaSO.transakcija = false;
              return OpstaSO.opsteIzvrsenjeSO(odo,p);
    }

   // Prekrivanje metode klase OpstaSO
   boolean izvrsenjeSO(OpstiDomenskiObjekat odo)
     { signal = BBP.nadjiSlogiVratiGa(odo);
           if (!signal)
              { BBP.dodajPorukuMetode("*Sistem ne moze da nadje " + odo.vratiNazivObjekta() + " po
                                                  zadatoj vrednosti.");
                return false;
              }
          BBP.dodajPorukuMetode("Sistem je nasao " + odo.vratiNazivObjekta() + " po zadatoj
                                              vrednosti.");
        return true;
    }
 }
```

5. Уговор UG5: *Storniraj*

Операција: Storniraj(*OpstiDomenskiObjekat*):signal;
Веза са СК: *СК3*
Предуслови: *Ако је доменски објекат сторниран не може се извршити системска операција.*
Постуслови: *Доменски објекат је сторниран*

```
class Storniraj extends OpstaSO
 {
    public static String Storniraj(OpstiDomenskiObjekat odo)
      { Storniraj st = new Storniraj();
        OpstaSO.transakcija = true;
       return OpstaSO.opsteIzvrsenjeSO(odo,st);
      }

    // Prekrivanje metode klase OpstaSO
   boolean izvrsenjeSO(OpstiDomenskiObjekat odo)
     { if (!Preduslov(odo))
            { BBP.dodajPorukuMetode("Sistem ne moze da stornira " + odo.vratiNazivObjekta() + "." +
                                              odo.vratiImeKlase() + " je vec storniran.");
              return false;
            }

      odo.Storniraj();

      if (!BBP.brisiSlog(odo))
            { BBP.dodajPorukuMetode("Sistem ne moze da stornira " + odo.vratiNazivObjekta() + ".");
              return false;
            }

      if (!BBP.pamtiSlozeniSlog(odo))
          { BBP.dodajPorukuMetode("Sistem ne moze da stornira " + odo.vratiNazivObjekta() + ".");
            return false;
          }

      BBP.dodajPorukuMetode("Sistem je stornirao " + odo.vratiNazivObjekta() + ".");
      return true;
 }
```

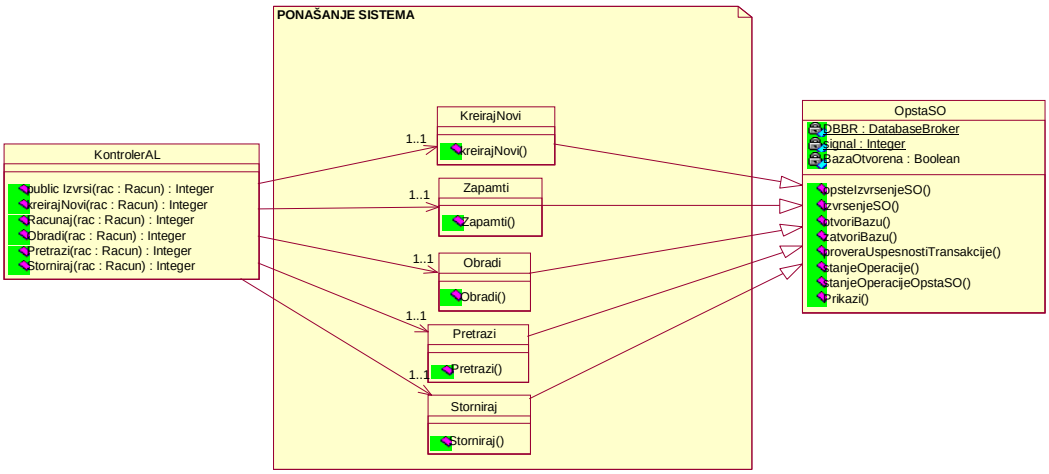
```
 private boolean Preduslov(OpstiDomenskiObjekat odo) // 94,77
   {  // 1. Ako je racun storniran nad njim se ne moze izvrsiti operacija racunanja stavki.
                        if  (BBP.vratiLogickuVrednostAtributa(odo,"Storniran") == true)
                          { return false; }
      return true;
   }
}
```

Након пројектовања сваке од СО прелази се на пројектовање класе која је одговорна за конекцију са базом и за контролу извршења трансакције. Метода која обезбеђује наведене захтеве се зове *opsteIzvrsenjeSO()*.

Свака од СО треба да наследи класу *OpstaSO* како би могла да се повеже са базом и како би се њено извршење пратило као трансакција:

```
class X extends OpstaSO
{
  public static String X(OpstiDomenskiObjekat odo)
    { X x = new X();
      OpstaSO.transakcija = true;
      return OpstaSO.opsteIzvrsenjeSO(odo,x);
    }
...
}
```

Klase koje su odgovorne za SO nasleđuju klasu OpstaSO (Slika OSO).



Slika OSO: Klase koje su odgovorne za SO nasleđuju klasu OpstaSO

Наводимо класу OpstaSO:

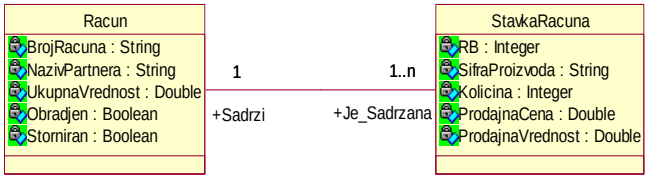
```
abstract class OpstaSO
{
  static BrokerBazePodataka BBP;
  static boolean signal;
  static boolean BazaOtvorena = false;
  static boolean transakcija = false;
  // Ova metoda je sinhronizovana kako bi se onemogućilo da 2 iil više klijenata u isto vreme izvršavaju
 // SO koja je pod transakcijom.
```

```
 synchronized static String opsteIzvrsenjeSO(OpstiDomenskiObjekat rac, OpstaSO os)
  { if (!os.otvoriBazu()) return os.vratiPorukuMetode();

     if (!os.izvrsenjeSO(rac) && transakcija)
          { signal = os.rollbackTransakcije();
            return os.vratiPorukuMetode();
           }

    if (transakcija) os.commitTransakcije();
       return os.vratiPorukuMetode();
  }

  abstract boolean izvrsenjeSO(OpstiDomenskiObjekat rac);


  boolean otvoriBazu()
  { if (BazaOtvorena == false)
         { BBP = new BrokerBazePodataka();
           BBP.isprazniPoruku();
          signal = BBP.otvoriBazu("RACUN");
          if (!signal) return false;
         }
    BBP.isprazniPoruku();
    BazaOtvorena = true;
    return true;
  }

  boolean commitTransakcije()
   { return BBP.commitTransakcije();}

  boolean rollbackTransakcije()
   { return BBP.rollbackTransakcije();}

  String vratiPorukuMetode()
   { System.out.println(BBP.vratiPorukuMetode());
     return BBP.vratiPorukuMetode();
   }
}
```

2.3.2.2.2 Пројектовање структуре софтверског система

На основу концептуалних класа праве се софтверске класе структуре (Слика АССКС).

Концептуалне класе:



Софтверске класе структуре:

```
class Racun
{
 String BrojRacuna;
 String NazivPartnera;
 Double UkupnaVrednost;
 boolean Obradjen;
 boolean Storniran;
 StavkaRacuna []sracun;

Racun()
  { BrojRacuna = "";
    NazivPartnera = "";
    UkupnaVrednost = new Double(0);
    Obradjen = false;
    Storniran = false;
    sracun = null;
  }

}

class StavkaRacuna
{ Integer RB;
  String SifraProizvoda;
  Integer Kolicina;
  Double ProdajnaCena;
  Double ProdajnaVrednost;
  Racun rac;

   StavkaRacuna(Racun rac1)
    { RB = new Integer(0);
      SifraProizvoda = new String("");
      Kolicina = new Integer(0);
      ProdajnaCena = new Double(0);
      ProdajnaVrednost = new Double(0);
      rac = rac1;
    }
}
```

2.3.2.3 Брокер базе података

Пре пројектовања датабасе брокера наводи се његова дефиниција и дефиниција свих оних концепата који су потребни да би се јасно схватила комуникација између датабасе брокера и базе података.

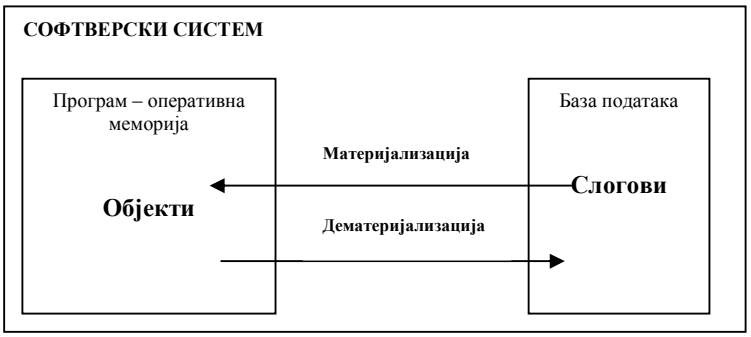
Дефиниција брокера базе података

Деф ПРПО1(Г. Бооцх): Објекат је **перзистентан** уколико настави да постоји и након престанка рада програма који га је створио.

Деф ПРПО2: Објекат је **перзистентан** уколико се може **материјализовати** и **дематеријализовати** (Слика МД).

Деф ПРПО3: Материјализација представља процес трансформације слогова из базе података у објекте програма[11].

Деф ПРПО4: Дематеријализација (Пасивизација) представља процес трансформације објеката из програма у слогове базе података.



Слика МД: Материјализација и дематеријализација објекта

Деф ПРПО5: Перзистентни оквир је скуп интерфејса и класа који омогућава перзистентност објектима различитих класа (Перзистентни оквир омогућава **перзистентни сервис**[12] објектима). Он се може проширити са новим интерфејсима и класама.

Деф ПРПО6: Перзистентни оквири су засновани на **Холивудском принципу**: *"Don't call us, we'll call you"*. То значи да кориснички дефинисане класе прихватају поруке од предефинисаних класа оквира.

Деф ПРПО7: Уколико у оквиру неке **трансакције**, операције мењају стање перзистентних објеката, оне не чувају то стање одмах у бази података. Уколико се жели запамтити стање које је настало као резултат извршених операција над перзистентним објектима позива се **commit операција**. Уколико се не жели запамтити стање које је настало као резултат извршених операција над перзистентним објектима позива се **roolback операција**. Commit и rollback операције представљају **трансакционе операције**.

Пример брокера базе података

У нашем примеру ми смо пројектовали перзистентни оквир (класа BrokerBazePodataka[13]) који ће да реализује следеће методе:

1. ***int otvoriBazu(String imeBaze)***
2. ***int commitTransakcije()***
3. ***boolean rollbackTransakcije()***
4. ***boolean pamtiSlog(OpstiDomenskiObjekat)***
5. ***boolean brisiSlog(OpstiDomenskiObjekat)***
6. ***boolean promeniSlog(OpstiDomenskiObjekat)***
7. ***boolean daLiPostojiSlog(OpstiDomenskiObjekat odo)***
8. ***boolean kreirajSlog(OpstiDomenskiObjekat)***
7. ***boolean nadjiSlogiVratiGa(Objekat,Objekat)***
8. ***boolean vratiLogickuVrednostAtributa(OpstiDomenskiObjekat odo, String nazivAtributa)***
9. ***boolean pamtiSlozeniSlog(OpstiDomenskiObjekat odo)***

[11] Термине материјализација и дематеријализација смо објаснили у ужем смислу као процесе који трансформишу објекте програма у слогове база података. У ширем смислу би се подразумевало да објекти могу бити сачувани у било ком перзистентном складишту података.

[12] Уколико **перзистентни сервис** омогућава памћење објеката у релационој бази података за њега се каже да је он **Object-Relation сервис пресликавања (mapping service).**

[13] Брокер базе података [Larman] патерн представља једну могућу реализацију перзистентног оквира. Брокер базе података [Larman] патерн је одговоран за материјализацију, дематеријализацију и кеширање објеката у меморији. Он се често назива и **Database Mapper** патерн.

Dajemo detaljno objašnjenje navedenih metoda.

```
class DatabaseBroker
{
  static Connection con;
  static Statement st;
```

/*Уговор DB1: **otvoriBazu**(String imeBaze) : boolean
Постуслов: Успостављена је веза (конекција) са базом података. Уколико је успешно остварена веза са базом података метода враћа вредност true и памти поруку: „*Uspostavljena je konekcija sa bazom podataka"*, иначе метода враћа false и памти поруке у зависности од следећих ситуација:
а) ако драјвер није учитан метода памти поруку: „Drajver nije učitan"
b) ако се десила грешка код конекције метода памти поруку: „Greška kod konekcije"
c) ако се десила грешка код заштите базе података. метода памти поруку: „Greška zaštite" */

```
public boolean otvoriBazu(String imeBaze)
  {  String Urlbaze;
     try {  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Urlbaze = "jdbc:odbc:" + imeBaze;
            con = DriverManager.getConnection(Urlbaze);
            con.setAutoCommit(false); // Ako se ovo ne uradi nece moci da se radi roolback.
          } catch(ClassNotFoundException e)
                { porukaMetode = "Drajver nije ucitan:" + e; return false;}
            catch(SQLException esql)
                { porukaMetode = "Greska kod konekcije:" + esql; return false;}
           catch(SecurityException ese)
                { porukaMetode = "Greska zastite:" + ese; return false;}
       porukaMetode = "Uspostavljena je konekcija sa bazom podataka."; return true;
  }
```

/*Уговор DB2: **commitTransakcije**() : boolean
Постуслов: Све операције које су мењале стање базе, од задњег позива commit или rollback transakcije, су успешно извршене (промене су успешно запамћене у бази). У том случају метода враћа true и памти поруку: „Uspešno urađen commit transakcije", иначе враћа false и памти поруку: "Nije uspešno urađen commit transakcije". */

```
 public boolean commitTransakcije()
  {  try { con.commit();
         } catch(SQLException esql)
                { porukaMetode = porukaMetode + "\nNije uspesno uradjen commit transakcije " + esql;
                  return false;
                }
    porukaMetode = porukaMetode + "\nUspesno uradjen commit transakcije ";
    return true;
}
```

/*Уговор DB3: **rollbackTransakcije**() : boolean
Постуслов: Ефекти свих операција које су мењале стање базе, од задњег poziva commit или rollback трансакције, су поништени (промене нису запамћене у бази). У том случају метода враћа true и поруку: "Uspesno uradjen rollback transakcije", иначе враћа false и поруку: „Nije uspešno uradjen rollback transakcije". */

```
public boolean rollbackTransakcije()
  {  try{ con.rollback();
         } catch(SQLException esql)
                { porukaMetode = porukaMetode + "\nNije uspesno uradjen rollback transakcije" + esql;
                  return false;
                }
    porukaMetode = porukaMetode + "\nUspesno uradjen rollback transakcije";
    return true;
  }
```

/*Уговор DB4: **pamtiSlog(OpstiDomenskiObjekat)**: boolean
Постуслов: Извршено је памћење домснског објекта у базу података (материјализација). Уколико је метода успешно извршена враћа труе и памти поруку: "Uspeštno zapamćen slog u bazi", иначе враћа false и памти поруку: „Nije uspešno zapamćen slog u bazi".
Напомена: Наведена метода је генеричка јер омогуцава памћење објеката било које класе у бази, уколико те класе наслеђује класу *OpstiDomenskiObjekat.*

```
public boolean pamtiSlog(OpstiDomenskiObjekat odo)
   {   String upit;
       try{ st = con.createStatement();
            upit = "INSERT INTO " + odo.vratiImeKlase()  +
                    " VALUES (" + odo.vratiVrednostiAtributa() + ")";
           st.executeUpdate(upit);
           st.close();
         } catch(SQLException esql)
              { porukaMetode = porukaMetode + "\nNije uspesno zapamcen slog u bazi. " + esql;
                return false;
              }
        porukaMetode = porukaMetode + "\nUspesno zapamcen slog u bazi. ";
       return true;
   }
```

/**
OBJAŠNJENJE POSTUPKA PROJEKTOVANJA GENERIČKE METODE
Уколико би метода била специфично везана за конкретну доменску класу нпр. Račun она би имала следећи изглед (болдовали смо зависне делове методе од доменске класе) :

```
public boolean pamtiSlog(Racun rac)
   {   String upit;
       try{ st = con.createStatement();
            upit = "INSERT INTO " + rac.vratiImeKlaseRacun()  +
                    " VALUES (" + rac.vratiVrednostiAtributaRacun() + ")";
           st.executeUpdate(upit);
           st.close();
         } catch(SQLException esql)
              { porukaMetode = porukaMetode + "\nNije uspesno zapamcen slog u bazi. " + esql;
                return false;
              }
            porukaMetode = porukaMetode + "\nUspesno zapamcen slog u bazi. ";
           return true;
   }

class Racun
{ ...
public String vratiVrednostiAtributaRacuna()
  { return ""+ BrojRacuna + "', '" + NazivPartnera + "', " +  UkupnaVrednost.doubleValue() + ", " + Obradjen + ", " +
    Storniran;}
public String vratiImeKlaseRacun() { return "Racun";}
}
```

Уколико би користили наведени приступ морали би за сваку доменску класу да имплементирамо методу *pamtiSLog(),* као и све остале методе које ћемо ниже навести (*brisiSlog(),promeniSlog(),...)* у класи BrokerBazePodataka. То није практично јер би број операција BrokerBazePodataka растао са појавом нових доменских класа. Због тога смо сваку од ниже наведених метода пројектовали као генеричку методу. На примеру методе *pamtiSlog()* објаснићемо општи поступак за пројектовање било које од ниже наведених генеричких метода.

Поступак пројектовања генеричке методе:
1. Одредити специфичне класе *(Racun)* у методи (*PamtiSlog*) која треба да постане генерална. Именовати специфичне класе у општем смислу (нпр. класу Račun са класом OpstiDomenskiObjekat). Такође методе специфичних класа именовати у општем смислу, ако је њихов назив повезан са нечим што је специфично за класу којој оне припадају (нпр. назив методе *vratiImeKlaseRacun()* са називом *VratiImeKlase()).*

```
public boolean pamtiSlog(OpstiDomenskiObjekat odo)
   {   String upit;
       try{ st = con.createStatement();
            upit = "INSERT INTO " + odo.vratiImeKlase()  +
                    " VALUES (" + odo.vratiVrednostiAtributa() + ")";
```

```
            st.executeUpdate(upit);
            st.close();
         } catch(SQLException esql)
                { porukaMetode = porukaMetode + "\nNije uspesno zapamcen slog u bazi. " + esql;
                  return false;
                }
     porukaMetode = porukaMetode + "\nUspesno zapamcen slog u bazi. ";
     return true;
   }
```

2. Направити генералну класу (апстрактну класу или интерфејс) за специфичне класе.

```
interface OpstiDomenskiObjekat
{ vratiVrednostiAtributa();
  vratiImeKlase(); }
```

Из наведеног може да се закључи да метода *pamtiSlog()*, може да прихвати различите доменске објекте преко параметра, ако доменски објекти наследе класу *OpstiDomenskiObjekat* и имплементирају њене методе *vratiIzraz() i vratiImeKlase()*.

У том смислу доменска класа Рацун и СтавкаРачуна ће добити следећи изглед:

```
class Racun implements OpstiDomenskiObjekat
{ ...
public String vratiVrednostiAtributa()
  { return """+ BrojRacuna + "', "' + NazivPartnera + "', " +  UkupnaVrednost.doubleValue() + ", " + Obradjen + ", " +
    Storniran;}
public String vratiImeKlase()
  { return "Racun";}
}

class StavkaRacuna implements OpstiDomenskiObjekat
{
 public String vratiVrednostiAtributa()
  { return """ + rac.BrojRacuna + "' , " + RB.intValue() + ", '" + SifraProizvoda + "', " +  Kolicina.intValue() + ", " +
ProdajnaCena.doubleValue() + ", " + ProdajnaVrednost.doubleValue();}

public String vratiImeKlase() { return "StavkaRacuna";}
}
```

Код доменских класа смо болдовали све делове програма који су промењени како би они могли да буду прихваћени као параметар методе *pamtiSlog()*.

КРАЈ ОБЈАШЊЕЊА ПОСТУПКА ПРОЈЕКТОВАЊА ГЕНЕРИЧКЕ МЕТОДЕ

```
/*************************************************************************************************
```

/*Уговор DB5: **brisiSlog**(OpstiDomenskiObjekat odo) : boolean
Постуслов: Обрисан је слог у бази података (такав објекат се не може више материјализовати). Уколико је метода успешно извршена враћа true и поруку: „Uspešno obrisan slog u bazi.“, иначе враћа false и памти поруку: “Nije uspešno obrisan slog u bazi.”.
Напомена: Наведена метода је генерицка јер омогуцава да се преко ње мозе обрисати објекат било које класе која наслеђује класу *OpstiDomenskiObjekat*.
/*

```
public boolean brisiSlog(OpstiDomenskiObjekat odo)
     {  String upit;
        try {  st = con.createStatement();
               upit ="DELETE *  FROM " + odo.vratiImeKlase() + " WHERE " + odo.vratiUslovZaNadjiSlog();
               st.executeUpdate(upit);
               st.close();
            } catch(SQLException esql)
                { porukaMetode = porukaMetode + "\nNije uspesno obrisan slog u bazi: " + esql;
                  return false;
                }
        porukaMetode = porukaMetode + "\nUspesno obrisan slog u bazi.";
        return true;
     }
```

/*Уговор DB6: **promeniSlog**(OpstiDomenskiObjekat odo) : boolean
Постуслов: Промењен је слог у бази података по задатом услову. Уколико је метода успешно извршена враћа true и памти поруку: "Uspešno promenjen slog u bazi podataka", иначе враћа false и памти поруку: "Nije uspešno promenjen slog u bazi podataka".
Напомена: Наведена метода је генерицка јер омогућава промену објекта било које класе која наслеђује класу *OpstiDomenskiObjekat.* /*

```
public boolean promeniSlog(OpstiDomenskiObjekat odo)
   {  String upit;
      try {  st = con.createStatement();
             upit =  "UPDATE " + odo.vratiImeKlase() +
                      " SET " + odo.postaviVrednostiAtributa() +
                      " WHERE " + odo.vratiUslovZaNadjiSlog();
            System.out.println("PROMENI SLOG" + upit);
            st.executeUpdate(upit);
            st.close();
           } catch(SQLException esql)
             { porukaMetode = porukaMetode + "\nNije uspesno promenjen slog u bazi podataka: " +
                                  esql;
               return false;
             }
        porukaMetode = porukaMetode + "\nUspesno promenjen slog u bazi podataka: ";
        return true;
    }
```

/*Уговор DB7: **daLiPostojiSlog**(OpstiDomenskiObjekat odo) : boolean
Постуслов: Уколико постоји слог у бази података метода враћа true и памти поруку: "Slog postoji u bazi podataka." иначе враћа false и памти поруку: „Slog postoji u bazi podataka."
Напомена: Наведена метода је генерицка јер омогућава претраживање објекта било које класе која наслеђује класу *OpstiDomenskiObjekat.* */

```
public boolean daLiPostojiSlog(OpstiDomenskiObjekat odo)
   {  String upit;
      ResultSet RSslogovi;
      try { st = con.createStatement();
            upit =  "SELECT *" +
                     " FROM " + odo.vratiImeKlase() +
                     " WHERE " + odo.vratiUslovZaNadjiSlog();
            RSslogovi = st.executeQuery(upit);
            boolean signal = RSslogovi.next();
            RSslogovi.close();
            st.close();
            if (signal == false)
             { porukaMetode = porukaMetode + "\nSlog ne postoji u bazi podataka.";
               return false; // Slog ne postoji u bazi.
             }
           } catch(SQLException esql)
               { porukaMetode = porukaMetode + "\nNije uspesno pretrazena baza: " + esql;
                 return false;
               }
       porukaMetode = porukaMetode + "\nSlog postoji u bazi.";
       return true;
     }
```

/*Уговор DB8: **kreirajSlog**(OpstiDomenskiObjekat odo): boolean
Постуслов: Креира нови слог у бази података метода. Уколико је метода успешно враћа true и памти поруку: "Slog postoji u bazi podataka." иначе враћа false и памти поруку: „Slog postoji u bazi podataka."
Напомена: Наведена метода је генеричка јер омогућава креирање објекта било које класе која наслеђује класу *OpstiDomenskiObjekat.* */

```
public boolean kreirajSlog(OpstiDomenskiObjekat odo)
    {  String upit;
       ResultSet rs;
       upit = "SELECT Max(" + odo.vratiAtributPretrazivanja()+ ") AS Max" +
              " FROM " + odo.vratiImeKlase();
        try { st = con.createStatement();
```

```
            rs = st.executeQuery(upit);


            if (rs.next() == false)
                odo.postaviPocetniBroj();
            else
                odo.povecajBroj(rs);

            upit = "INSERT INTO " + odo.vratiImeKlase()  +
                    " VALUES (" + odo.vratiVrednostiAtributa() + ")";
            st.executeUpdate(upit);
            st.close();
            } catch(SQLException esql)
              {  porukaMetode = porukaMetode + "\nNe moze da se kreira novi slog: " + esql;
                 return false;
              }
          porukaMetode = porukaMetode + "\nKreiran je novi slog: ";
          return true;
    }
```

/*Уговор DB9: **nadjiSlogiVratiGa**(OpstiDomenskiObjekat odo, OpstiDomenskiObjekat odo1) : integer
/*

```
public boolean nadjiSlogiVratiGa(OpstiDomenskiObjekat odo)
      {  ResultSet RS;
         String nazivVezanogObjekta;
         int brojStavki;
        String upit;
        Statement st;
         try {
                st= con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_READ_ONLY);
                upit = "SELECT *" +   " FROM " + odo.vratiImeKlase() +
                       " WHERE " + odo.vratiUslovZaNadjiSlog();
               RS = st.executeQuery(upit);
               System.out.println("Upit-nadji: " + upit);
               boolean signal = RS.next();
              if (signal == false)
                 { porukaMetode = porukaMetode + "\nNe postoji slog u bazi podataka.";
                   return false;
                 }
               porukaMetode = porukaMetode + "\nUspesno je procitan slog iz baze podataka.";

              if (odo.Napuni(RS))  // if1
                  { for (int j=0;j<odo.vratiBrojVezanihObjekata();j++)
                        { OpstiDomenskiObjekat vezo = odo.vratiVezaniObjekat(j);
                          if (vezo == null)
                              { porukaMetode = porukaMetode + "\nNe postoji vezani objekat a navedeno je
                                  da postoji.";
                                return false;
                              }
                          else  // else1
                             { upit = "SELECT COUNT(*) as brojStavki" +   " FROM " + vezo.vratiImeKlase() +
                                      " WHERE " + vezo.vratiUslovZaNadjiSlogove();
                               RS = st.executeQuery(upit);
                               if (RS.next() == false)
                                  { porukaMetode = porukaMetode + "\nNe postoje slogovi vezanog objekta";
                                    return true;
                                  }
                             brojStavki = RS.getInt("brojStavki");
                             odo.kreirajVezaniObjekat(brojStavki,j);
                             upit = "SELECT *" +  " FROM " + vezo.vratiImeKlase() +
                                    " WHERE " + vezo.vratiUslovZaNadjiSlogove();
                             RS = st.executeQuery(upit);
                             int brojSloga = 0;
                             while(RS.next())
                                   { odo.Napuni(RS,brojSloga,j);
                                     brojSloga ++;
```

```
                        }
                    porukaMetode = porukaMetode + "\nUspesno su procitani slogovi vezanog objekta";
                     } // end else1
                   } // end for
               } // end if1
          RS.close();
          st.close();
             } catch(Exception e)
                 { porukaMetode = porukaMetode + "\nGreska kod citanja sloga iz baze podataka." + e;
                   return false;
                 }
          return true;
      }/*
```

/*Уговор DB10: **vratiLogickuVrednostAtributa**(OpstiDomenskiObjekat odo, String nazivAtributa) : boolean
/*

```
public  boolean vratiLogickuVrednostAtributa(OpstiDomenskiObjekat odo, String nazivAtributa)
  { String upit;
    ResultSet rs;
     boolean s = false;
     try { st = con.createStatement();
          upit = " SELECT *" +
                  " FROM " + odo.vratiImeKlase() +
                  " WHERE " + odo.vratiUslovZaNadjiSlog();
          System.out.println("upit: " + upit);
          rs = st.executeQuery(upit);
          rs.next();
          s = KonverterTipova.Konvertuj (rs, s, nazivAtributa);
          rs.close();
          st.close();
         } catch(Exception e)
           {  porukaMetode = porukaMetode + "\nGreska kod citanja logicke vrednosti atributa sloga." + e;
              return false;
           }
       return s;
     }
```

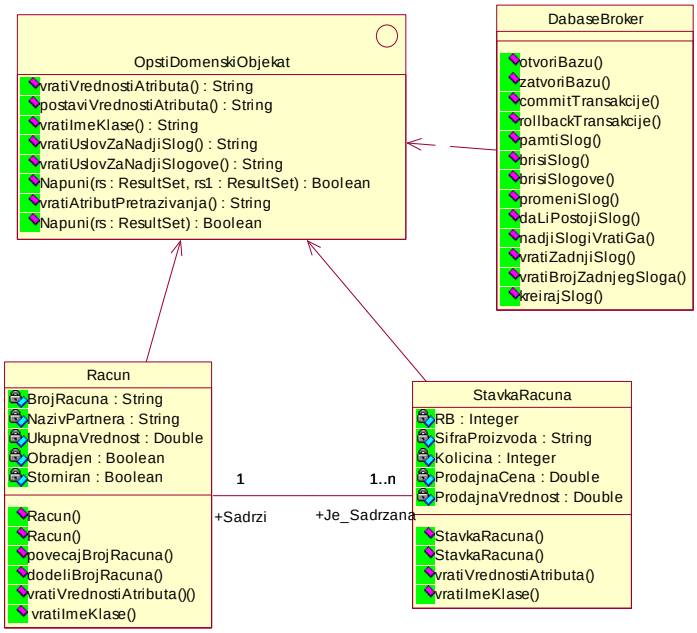
Ugovor DB11: **pamtiSlozeniSlog(OpstiDomenskiObjekat odo):** boolean

```
  public boolean pamtiSlozeniSlog(OpstiDomenskiObjekat odo)
    { String upit;
      try { st = con.createStatement();
            upit = " INSERT INTO " + odo.vratiImeKlase()  +
                    " VALUES (" + odo.vratiVrednostiAtributa() + ")";
          st.executeUpdate(upit);
          for(int j=0;j<odo.vratiBrojVezanihObjekata();j++)
              { OpstiDomenskiObjekat vezo;
                for(int i=0; i < odo.vratiBrojSlogovaVezanogObjekta(j);i++)
                    { vezo = odo.vratiSlogVezanogObjekta(j,i);
                      upit = " INSERT INTO " + vezo.vratiImeKlase()  +
                              " VALUES (" + vezo.vratiVrednostiAtributa() + ")";
                       st.executeUpdate(upit);
                    }
              }

          st.close();
         } catch(SQLException esql)
            {  porukaMetode = porukaMetode + "\nNije zapamcen slozeni slog: " + esql;
               return false;
            }
       porukaMetode = porukaMetode + "\nZapamcen je slozeni slog.";
       return true;
     }
```

U procesu pravljenja generičkih metoda DatabaseBroker klase dobili smo metode interfejsa OpstiDomenskiObjekat:

```
// Operacije navedenog interfejsa je potrebno da implementira svaka od domenskih klasa,
// koja zeli da joj bude omogucena komunikacija sa Database broker klasom.

interface OpstiDomenskiObjekat
{ String vratiVrednostiAtributa();
 String postaviVrednostiAtributa();
 String vratiImeKlase();
 String vratiUslovZaNadjiSlog();
 String vratiUslovZaNadjiSlogove();
 boolean Napuni(ResultSet rs, ResultSet rs1);
 String vratiAtributPretrazivanja();
 boolean Napuni(ResultSet rs);
}
```

Као резултат пројектовања класе DatabaseBrokera и interfejsa OpstiDomenskiObjekat dobijaki se sledeći dijagrami klasa (Slika DBBR, ASSDBBR)



Slika DBBR: Database broker klasa se povezuje sa klasom OpstiDomenskiObjekat

2.3.3 Пројектовање складишта података

На основу софтверских класа структуре пројектовали смо табеле (складишта података) релационог система за управљање базом података (Слика АССТБП).

Table: Racun

Columns

Name	Type	Size
BrojRacuna	Text	10
NazivPartnera	Text	50
UkupnaVrednost	Number (Double)	8
Obradjen	Yes/No	1
Storniran	Yes/No	1

Table Indexes

Name	Number of Fields
PrimaryKey	1
Fields:	BrojRacuna, Ascending

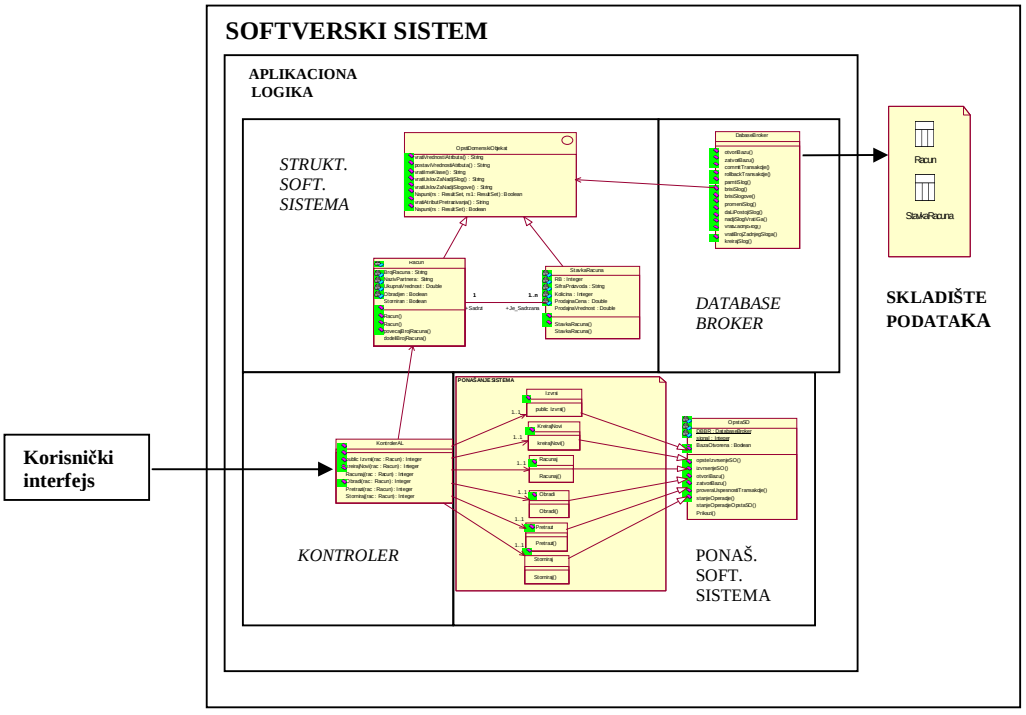
Table: StavkaRacuna

Columns

Name	Type	Size
BrojRacuna	Text	10
RB	Number (Integer)	2
SifraProizvoda	Text	20
Kolicina	Number (Integer)	2
ProdajnaCena	Number (Double)	8
ProdajnaVrednost	Number (Double)	8

Table Indexes

Name	Number of Fields
PrimaryKey	**2**
Fields:	BrojRacuna, Ascending RB, Ascending



Слика АССТБП: Архитектура софт. система након пројектовања табела базе података