УНИВЕРЗИТЕТ „УНИОН“
РАЧУНАРСКИ ФАКУЛТЕТ
Кнез Михаилова 6/VI
11000 БЕОГРАД

Број:
Датум:

УНИВЕРЗИТЕТ УНИОН
РАЧУНАРСКИ ФАКУЛТЕТ
БЕОГРАД
Програмирање

ДИПЛОМСКИ РАД

Кандидат: **Срђан Обућина**
Број индекса: **97/05**

Тема рада: **Брз развој апликација – Систем за учење на даљину**

Ментор рада: **проф. др. Стеван Милинковић**

Београд, 2010.

Садржај

1 Увод

Дипломски рад описује модел брзог развоја апликација и приказује његову употребу у развоју система за учење на даљину, са ограничењем на први циклус развоја.

Модел брзог развоја апликација (Rapid Application Development, RAD) је модел развоја софтвера који се фокусира на развој апликација у кратком временском року, обично уз компромисе у употребљивости, имплементираним функционалностима или брзини извршавања. Модел се ослања на технике израде прототипова, цикличног прилагођавања и интензивну употребу софтверских компоненти и алата за пројектовање уз помоћ рачунара. У овом моделу имплементација има највећи значај, планирање се своди само на неопходни минимум, функционалности се обликују према могућностима постојећих библиотека и софтверских компоненти, врло често се објављују пробне верзије и консултују корисници.

Систем за учење на даљину се реализује као *Мајкрософт Виндоуз* (Microsoft Windows) клијент-сервер апликација са аудио и видео стриминигом у реалном времену. За пројектовање се користи *Моделмејкер* (Modelmaker), а за израду *Ембаркадеро РАД Студио* (Embarcadero RAD Studio), програмски језик *Објектни Паскал* (Object Pascal), помоћни алати и софтверске компоненте неколико произвођача.

Серверски део система је задужен за проверу идентитета корисника, управљање информацијама о активним сесијама и управљање саобраћајем од и ка корисницима. Серверски део прима податке од клијената (аудио и видео сигнале и снимак екрана) и прослеђује их другим клијентима. Серверске команде се преносе ТЦП, а аудио и видео сигнали УДП протоколом.

Клијентски део система омогућава корисницима да започну ново предавање или да прате неко које је тренутно у току. Садржи неколико интегрисаних компоненти које се користе за презентацију наставног материјала - ПДФ читач, текстуални едитор са бојењем синтаксе, веб читач и табла за цртање и писање са препознавањем рукописа. Корисник који је започео предавање је предавач, и може да приказује слајдове другим корисницима и помоћу графичке табле са оловком да црта и пише или да означава важне делове на слајдовима.

2 Модел брзог развоја апликација

2.1 Процес развоја софтвера

Процес развоја софтвера је структурирани скуп активности које се спроводе током израде софтверског производа. Циљ овог процеса је да се у оквиру планираних временских и материјалних ограничења направи квалитетан софтверски производ, који у потпуности задовољава захтеве корисника и једноставан је за одржавање и проширивање.

У општем случају, процес развоја софтвера се састоји из следећих фаза:

1. Анализа и дефинисање захтева
2. Пројектовање система
3. Пројектовање програма
4. Имплементација програма
5. Тестирање јединица
6. Интегративно тестирање
7. Тестирање система
8. Испорука система
9. Одржавање

Активности у процесу развоја софтвера спроводи развојни тим састављен од софтверских инжењера специјализованих за појединачне фазе процеса. У табели су приказане улоге чланова развојног тима у различитим фазама процеса развоја софтвера.

Члан развојног тима	Аналитичар	Пројектант	Програмер	Тестни инжењер	Предавач
Фаза развоја					
Анализа и дефинисање захтева	■				
Пројектовање система	■	■			
Пројектовање програма		■	■		
Имплементација програма			■		
Тестирање јединица			■	■	
Интегративно тестирање				■	
Тестирање система				■	
Испорука система					■
Одржавање	■	■	■	■	■

Задатак аналитичара је да сакупе жеље корисника и формулишу их у појединачним захтевима од којих се формира спецификација захтева. Спецификација захтева описује

функционисање система са становишта корисника и један је од кључних докумената у процесу развоја софтвера.

Разматрајући спецификацију захтева, аналитичари и пројектанти састављају пројекат система, који се бави избором најбољег решења за дати проблем. У пројекту система се предлажу архитектура система, ниво апстракције и модуларности компоненти, елементи корисничког интерфејса, начин обраде изузетака и грешака и слично.

Током фазе пројектовања програма пројектанти раде са програмерима на представљању система на начин који омогућава програмерима да имплементирају оно што је формулисано у захтевима. Резултат ове фазе је пројекат програма, који описује ентитете и везе између њих које треба имплементирати да би се добила тражена функционалност.

У фази имплементације, програмери имплементирају функционалности по предложеном пројекту програма, користећи се технологијама које им омогућавају да на најбољи начин изађу у сусрет постављеним захтевима.

Имплементиране функционалности је потребно тестирати. Тестирање се изводи у више корака. Иницијалне тестове компоненти ниског нивоа раде сами програмери. Тестни инжењери праве планове тестирања по којима ће, током фаза интегративног тестирања и тестирања система, проверавати да ли функционалне групе и систем у целини раде у складу са спецификацијом захтева.

Када се систем испоручи предавачи организују обуку за оперативно коришћење система. У ту сврху могу припремити предавања, слајдове, кратке филмове и друге сличне материјале, помоћу којих ће упознати корисника са свим аспектима функционисања система који су релевантни за његову примену у пракси.

Током фазе одржавања, чланови развојног тима пружају техничку подршку корисницима и од њих прикупљају податке о уоченим проблемима у раду, могућим унапређењима и новим захтевима. Из фазе одржавања се могу поново покренути неке од фаза процеса развоја, да би се исправили проблеми уочени у раду или изашло у сусрет новим жељама и захтевима корисника.

У идеалном случају процес развоја софтвера се одвија секвенцијално, а софтверски производ је завршен када се прођу све фазе процеса. Међутим, у пракси се неретко јављају околности због којих није могуће одржати редослед извођења активности или се неке од њих морају понављати. На пример, током пројектовања система, може се открити да неки захтеви нису документовани или купац може променити своје захтеве. За време имплементације или тестирања система може се показати да неки делови не раде у складу са захтевима, па их је потребно исправити, поново пројектовати програм или систем или поново преговарати са корисником о решењу конкретног проблема. Исто тако, у почетним фазама оперативне употребе, купац може схватити да неке функционалности не излазе у сусрет његовим потребама на најбољи начин, па пожели да их унапреди, осмисли потпуно нове или чак открити да му нека функционалност коју је првобитно тражио уопште није потребна. Сваки од ових догађаја узрокује напуштање праволинијског проласка кроз фазе процеса развоја софтвера и потребу за враћањем на неку од претходних фаза.

2.2 Модел развоја софтвера

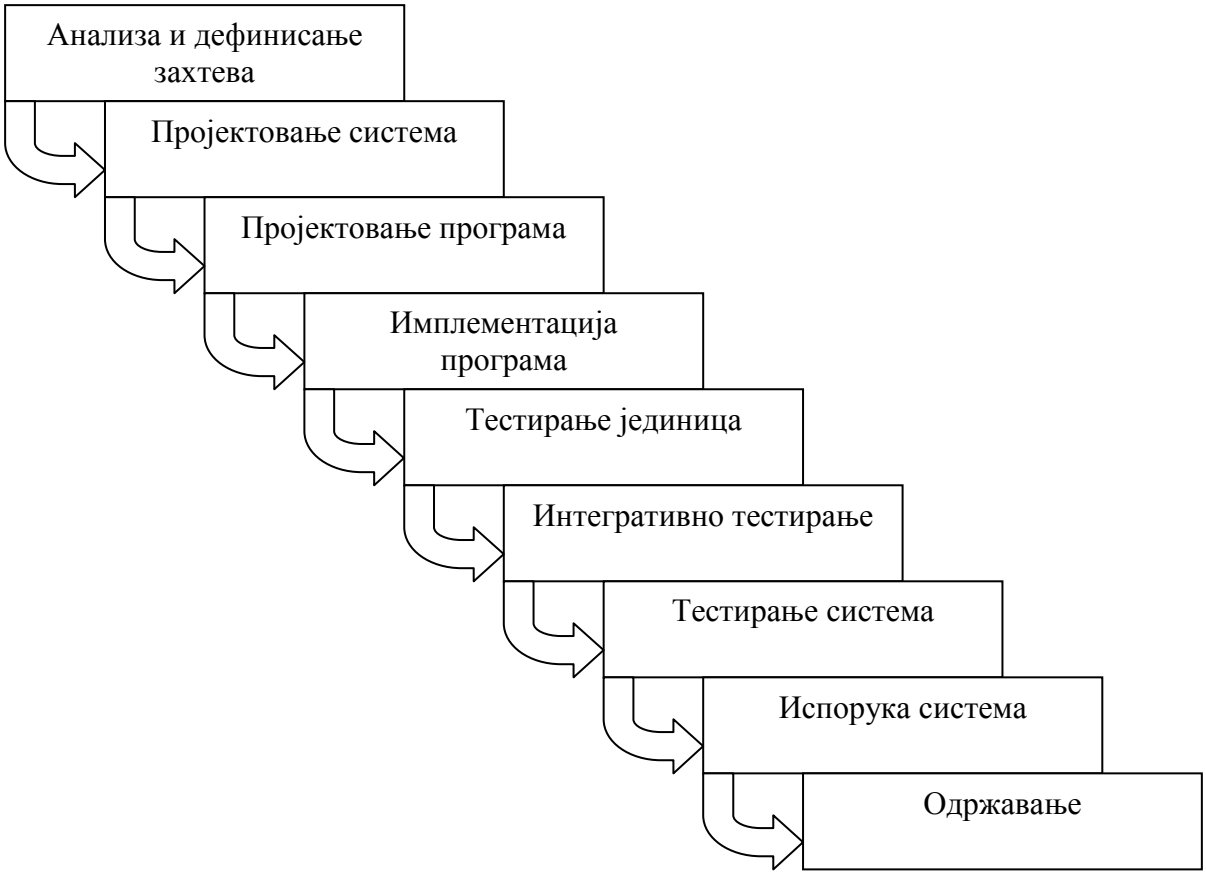
Модел развоја или методологија развоја има задатак да структурира и контролише процес развоја софтвера. Модел развоја одређује кроз које фазе процеса развоја софтвера и којим редоследом треба проћи и колико времена треба посветити којој фази. Њиме се у процес могу додати и друге посебне фазе, али и изоставити нека од наведених, или захтевати да се неке фазе понављају више пута. Модел развоја може одређивати алате и технологије које ће се користити у процесу развоја, као и састав развојног тима и улоге у њему.

Данас постоји релативно велики број модела развоја софтвера, али ни један од њих није универзално применљив за сваки софтверски производ. Различити модели су намењени различитим врстама софтверских производа и решавају различите организационе и техничке проблеме. Један модел може стављати акценат на правилно и детаљно планирање, а други на развој производа по функционалним компонентама или на рад под временским и материјалним ограничењима. Неки модели су детаљна упутства како треба спроводити процес развоја софтвера, а неки су само његови мање или више уопштени описи и дозвољавају већу слободу у самом спровођењу процеса. Правилан избор модела развоја је од кључног значаја за успешност испуњавања постављеног плана развоја софтверског производа.

У наставку ћемо се осврнути на неке од модела и техника развоја чији су елементи уграђени у модел брзог развоја апликација и анализирати њихове предности и мане, а затим и детаљније описати модел брзог развоја апликација.

2.3 Модел водопада

Модел водопада је најстарији и најједноставнији модел развоја. У моделу водопада, кроз фазе развоја се пролази секвенцијално, нема преклапања фаза и нема итерација кроз фазе. На следећу фазу се прелази тек када се претходна фаза у потпуности заврши.



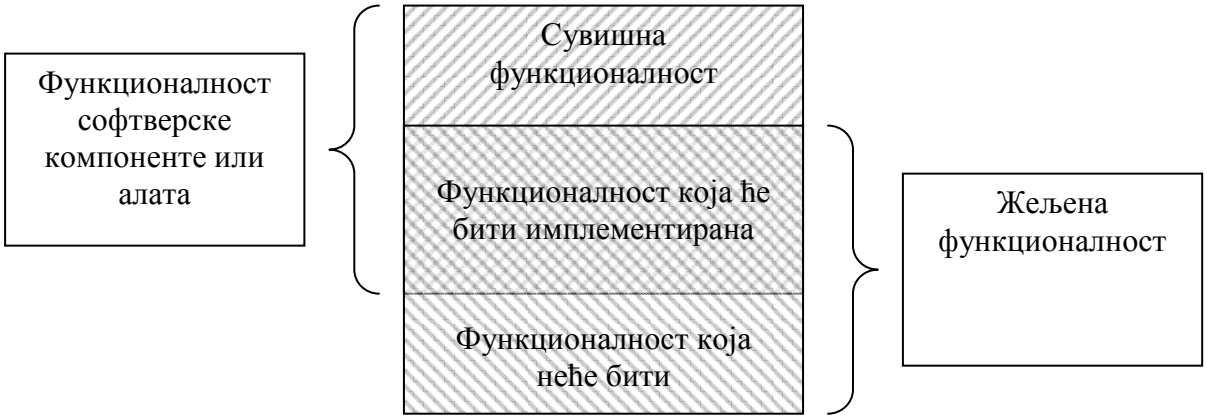
Слика 1 - Модел водопада

Модел водопада је веома зрео, постоји око 40 година и користио се у развоју великог броја различитих врста софтвера. Добро се показује у пројектима са сложеном али потпуно јасном структуром, пошто омогућава систематично разлагање сложености. Добро се показује и када ниски трошкови развоја немају приоритет над квалитетом и временом развоја или када развојни тим није довољно обучен или му недостаје искуства. Пошто су границе фаза развоја јасне и недвосмислене, модел је предвидљив и даје веома јасну слику о томе докле је одмакла реализација софтверског производа. Модел захтева да се систем комплетно и квалитетно документује, што значајно олакшава одржавање.

Моделу водопада се замера нефлексибилност зато што захтева да се цео посао израде софтвера изведе у једном покушају и зато што не може да одговори на промене које могу да настану за време израде софтера. Документација је кључна и критична ставка комуникације између чланова тима, а за израду квалитетне документације је потребно доста времена. Некомплетна или неквалитетна документација битно отежава рад у наредној фази процеса. Пошто захтева да се једна фаза у потпуности заврши пре него што почне следећа, модел водопада може значајно продужити време испоруке, јер доводи до ситуације да се секвенцијално обављају послови који се могу обављати паралелно. На пример, ако је за неке делове система завршена фаза пројектовања и не постоје препреке да се пређе на имплементацију тих делова, модел водопада налаже да се то не чини.

2.4 Пројектовање по алатима

Пројектовање по алатима је радикалан приступ који се у прошлости користио само под изузетно екстремним временским ограничењима. Данас, са напретком развојних окружења и софтверских компоненти, техника пројектовања по алатима је све више применљива и под другим околностима. Техника се користи у фазама пројектовања система и пројектовања програма, а може се користити као додатак скоро сваком постојећем моделу развоја.



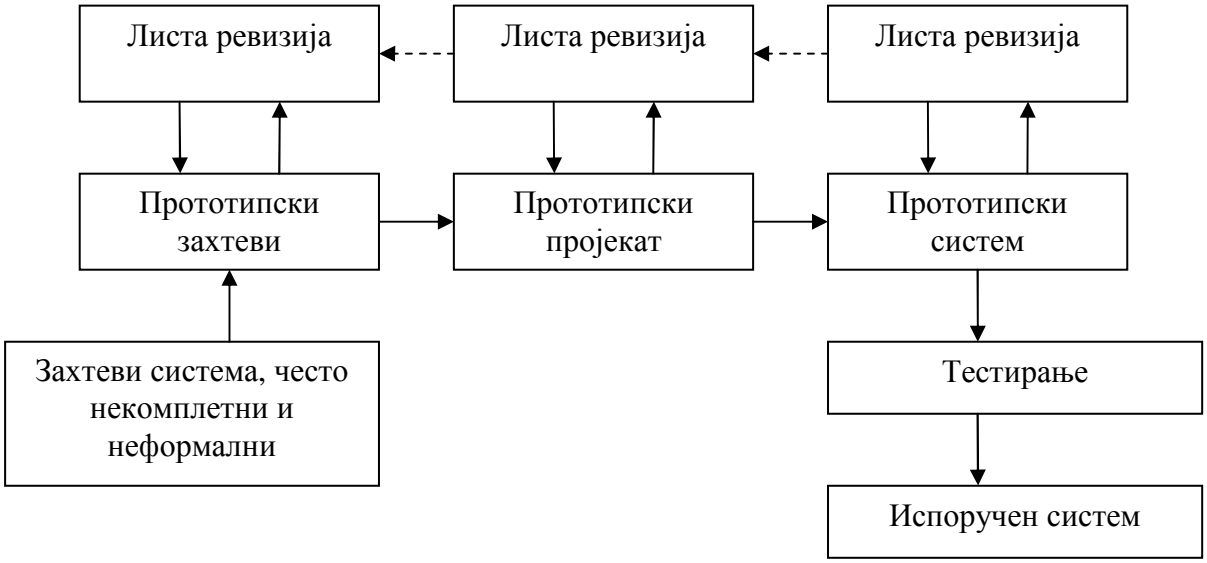
Слика 2 - Пројектовање по алатима

Основна идеја ове технике је да се нека функционалност софтверског производа имплементира само ако је директно подржава нека постојећа софтверска компонента или алат. Слика сугерише да је извесно да нека функционалност не мора бити имплементирана на жељени начин. У односу на планирану функционалност софтверског производа, овај недостатак може ићи у оба смера. Софтверска компонента или алат може нудити непотпуну функционалност или функционалност која превазилази планирану функционалност софтверског производа. Ипак, уз правилан избор софтверске компоненте, алата и пажљиво пројектовање, може се постићи највећи део планиране функционалности.

Пројектовање по алатима остварује највећу могућу брзину развоја. Готов софтверски производ се добија једноставним склапањем постојећих софтверских компоненти. Али, техника има и неколико озбиљних недостатака. Развојни тим у извесној мери губи контролу над производом, јер није у могућности да имплементира све функционалности које жели, нити на начин на који жели. Производ постаје зависан од производа других произвођача, њихових стратегија развоја и финансијске стабилности, шти модел чини непогодним за софтверске производе који ће захтевати дуготрајну примену и одржавање.

2.5 Прототипски модел

Прототипски модел подразумева израду прототипа – некомплетних делова система, који се брзо развију, а затим анализирају са циљем да се дође до бољег разумевања отворених питања и смање неодређености у даљем пројектовању. Развој почиње номиналним скупом захтева, а затим се истражују алтернативе тако што се анализирају могући елементи програма или излази неког већ постојећег система. Када корисник дефинише своје жеље, долази до ревизије захтева. Када се постигне договор о захтевима, прелази се на пројектовање. Затим се истражују алтернативе у пројекту, који се ревидира док се не добије задовољавајући резултат. Ревизија пројекта може открити грешке у захтевима, па је омогућен и повратак на претходну фазу. Најзад, пише се изворни код, поново уз истраживање алтернатива, и по добијању задовољавајућег резултата, систем се тестира и испоручује.

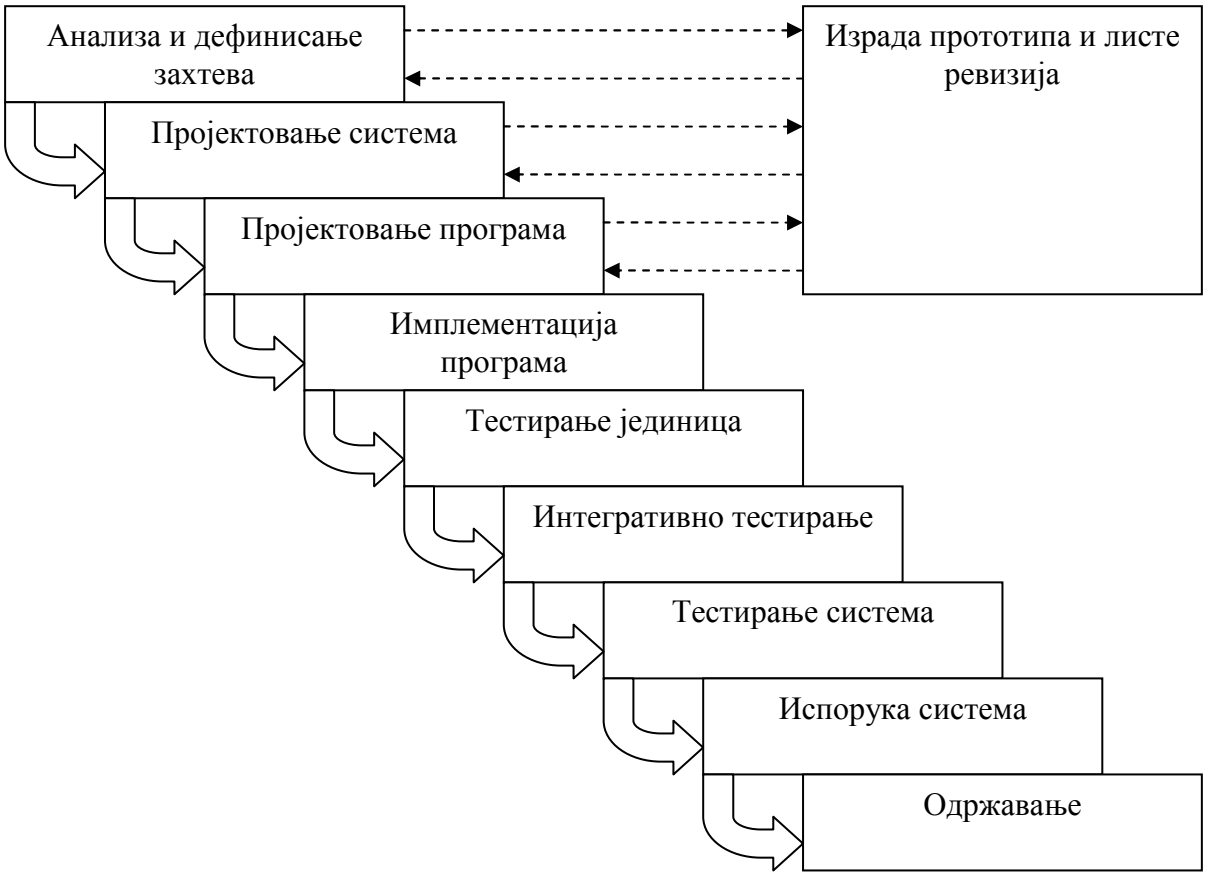


Слика 3 - Прототипски модел

Неке предности прототипског модела је што се већ у најранијим фазама развоја проналазе недоследности и што се дозвољава корисницима да процене решења која им нуди развојни тим. Прототипски модел омогућава да се избегну велики трошкови и избегну тешкоће измена већ готовог софтверског производа. Изузетно је користан код развоја корисничког интерфејса.

Неки недостаци су недовољна анализа система као целине услед фокусирања на делове система за које се израђује прототип и потенцијално дуже време и виши трошкови развоја.

Прототипски модел се не мора користити самостално, као комплетан модел. Могуће га је користити само као технику у оквиру других модела развоја. На слици је приказан прототипски модел као додатак моделу водопада.

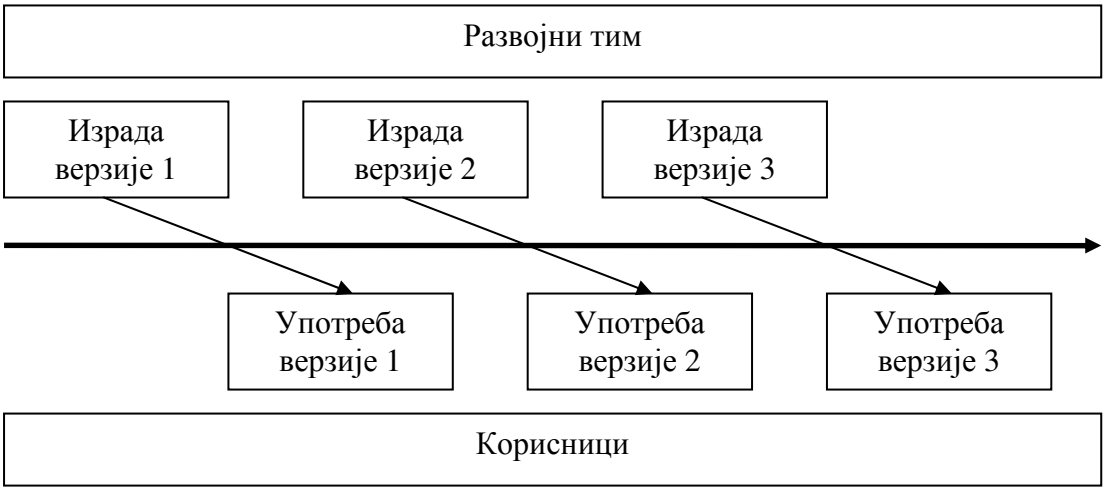


Слика 4 - Прототипски модел као додатак моделу водопада

2.6 Циклични развој: Инкрементални и итеративни развој

Велики број модела развоја се бави софтверским производом као једном недељивом целином. Другим речима, потребно је за софтверски производ у целини анализирати и дефинисати захтеве, пројектовати га, израдити и тестирати. Када дође до испоруке софтверског производа, он је комплетан и испуњава све захтеве које је поставио корисник. Проблем са оваквим приступом је што време које протекне од почетка до краја развоја може постати неприхватљиво дуго. Данашња пословна окружења се мењају изузетно брзо и од софтвера се захтева да истом брзином излази у сусрет тим променама.

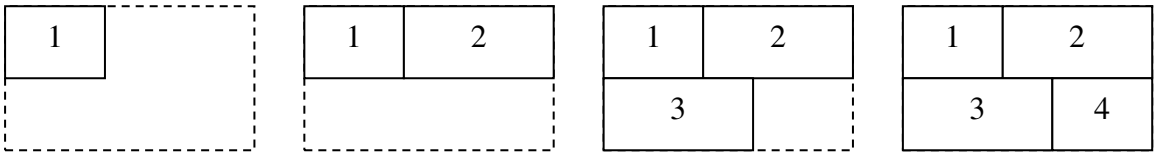
У цикличном развоју, софтверски производ се пројектује тако да може да се испоручује у деловима, чиме се корисницима омогућава да имају део функционалности док је остатак још у развоју. Испоручени систем се означава верзијом и обично постоје бар две верзије. Једна верзија је пуштена у оперативну употребу, а друга је развојна верзија, на којој ради развојни тим. Свака наредна верзија треба на неки начин да унапреди претходну.



Слика 5 - Циклични развој и верзије софтвера

Два најпознатија приступа развоју софтверског производа по верзијама су су инкрементални и итеративни развој.

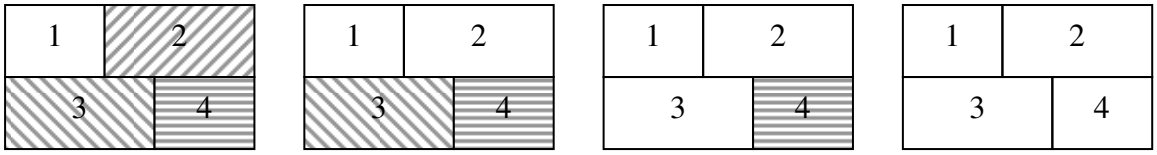
У инкременталном развоју, систем дефинисан у спецификацији захтева се дели на подсистеме према функцијама. Верзије се дефинишу као мали функционални подсистеми, а затим се у свакој новој верзији додају нове функције. Кроз одређен број верзија, долази се до система са потпуном функционалношћу.



Слика 6 - Инкрементални развој

У итеративном развоју, на почетку се испоручује систем са свим функционалностима, али оне могу имати недостатке који не утичу битно на исправан рад. На пример, неки алгоритам није имплементиран на најбољи могући начин, нека операција има сувишне кораке или је кориснички интерфејс тренутно недовољно интуитиван. У свакој новој

верзији мењаће се функционалности подсистема и приближавати се потпуно функционалном систему у складу са спецификацијом захтева.



Слика 7 - Итеративни развој

Циклични развој омогућава веома рани контакт корисника са системом. Развојни тим може да прати корисников начин употребе система и да на основу тог посматрања дефинише побољшања у пројекту система или пројекту програма за касније верзије. Честе верзије омогућавају брзо исправљање уочених проблема у продукционом систему. Развојни тим се може усредсредити на унапређење различитих аспеката система у различитим верзијама. На пример, у једној верзији може радити на унапређењу корисничког интерфејса, а у следећој на ефикаснијој употреби рачунарских ресурса.

У литератури се израз „итеративни развој“ доста често користи да означи циклични развој. Иако су у контексту природних језика семантички слични, ови изрази у контексту процеса развоја софтвера имају веома различита значења. Циклични развој је шири појам од итеративног развоја и означава сваки развој током кога долази до вишеструког проласка кроз фазе процеса развоја. Итеративни развој је само један приступ цикличном развоју. Због овога је потребно посебно обратити пажњу на контекст у коме се изрази користе, да би се правилно разумело на шта тачно аутор мисли.

2.7 Модел брзог развоја апликација

2.7.1 Увод

Традиционални модели развоја, а нарочито модел водопада, су често имали за последицу да се развије софтвер који не испуњава актуелне захтеве корисника. Дешавало се да развој траје толико дуго да дође до промене захтева пре него што је систем завршен. За овај проблем су понуђена решења у виду разних модела и техника развоја, а једно од њих је и модел брзог развоја апликација. Крајем осамдесетих година двадесетог века, *Скот Шулц* (Scott Shultz) и *Џејмс Мартин* (James Martin), комбинујући идеје израде прототипова и цикличног развоја, дефинисали су модел брзе итеративне израде прототипа (Rapid Iterative Production Prototyping, RIPP) која се фокусира на развој система у кратком временском року. *Џејмс Мартин* касније проширује и формализује овај модел, и 1991. објављује књигу Брз развој апликација (Rapid Application Development, RAD).

> Џејмс Мартин је рођен 1933. у Енглеској. Уз виши докторат (Litt.D.) Оксфорд универзитета, има и почасне докторате на свих шест континената. Специјалиста је у области пројектовања система, модела развоја софтвера и информатичког инжењеринга. Многи га сматрају оцем пројектовања уз помоћ рачунара (CASE). Први прототипови алата за пројектовање уз помоћ рачунара компанија *Тексас Инструментс* (Texas Instruments) и *Нолиџвер* (KnowledgeWare), су прављени у његовом дому и по његовим упутствима. Један је од првих промотера програмских језика четврте генерације. Написао је 102 књиге, од којих су неке радови који су изменили схватања у индустрији информационих технологија. У јубиларном издању за своју 25. годишњицу, часопис *Компјутерворлд* (Computerworld) га је прогласио четвртим најутицајнијим човеком у области рачунарских наука.

Модел брзог развоја апликација (Rapid Application Development, RAD) је модел развоја софтвера који се фокусира на развој апликација у кратком временском року, обично уз компромисе у употребљивости, имплементираним функционалностима или брзини извршавања. Модел се ослања на технике израде прототипова, цикличног прилагођавања и интензивну употребу софтверских компоненти и алата за пројектовање уз помоћ рачунара. Скраћеница *RAD* се данас често користи и као заједнички назив за све моделе и технике развоја који су фокусирани на брзину развоја и за описивање намене модерних развојних окружења.

2.7.2 Кључни елементи модела брзог развоја апликација

2.7.2.1 Пројектовање по алатима

Приликом пројектовања система и програма, кад год је могуће, функционалности се пројектују према могућностима постојећих софтверских компоненти и алата, чак и ако то значи нарушавање функционалности у мањем обиму или повећање цене израде због издатака за куповину неке софтверске компоненте.

2.7.2.2 Алати за брз развој

Модел брзог развоја апликација подразумева интензивну употребу алата који убрзавају развој, попут алата за пројектовање уз помоћ рачунара, алата за моделовање захтева, алата за моделовање података, развојних окружења прилагођених брзом развоју, генератора изворног кода, генератора документације и других.

2.7.2.3 Циклични развој

У цикличном развоју, свака наредна верзија је функционалнија у односу на претходну. Циклуси се понављају све док се не имплементирају све предвиђене функционалности. У сваком циклусу корисник има прилику да се упозна са новим функционалностима, изнесе своје утиске и дефинише нове или прилагоди постојеће захтеве. Увидом у имплементиране функционалности корисник има и јасну представу о томе колико је развој одмакао.

2.7.2.4 Израда прототипова

Прототипови се израђују у сврху убрзавања пројектовања и прикупљања захтева корисника. Иницијално се прави једноставан прототип, који се демонстрира као концептуално решење и који постаје полазна тачка за даља размишљања о захтевима и пројекту софтвера. Брз развој прототипова се постиже употребом алата за пројектовање помоћу рачунара, израдом модела података, конверзијом модела података у базу података и употребом алата за генерисање изворног кода.

2.7.2.5 Временска ограничења

У брзом развоју време је приоритетна категорија, па време испоруке има предност над функционалностима система. Ако нека функционалност не може да се имплементира у планираном временском року, имплементација се одлаже за следећу верзију. Временска ограничења су важна зато што би евентуалне неконтролисане измене, узроковане недоследностима у захтевима, превиђањем неког захтева, недовољно планираном дизајну, лошом организацијом и сл, могле значајно продужити време развоја, чиме се одлажу и демонстрација прототипа и контакт са корисницима и прикупљање информација неопходних за даљи развој.

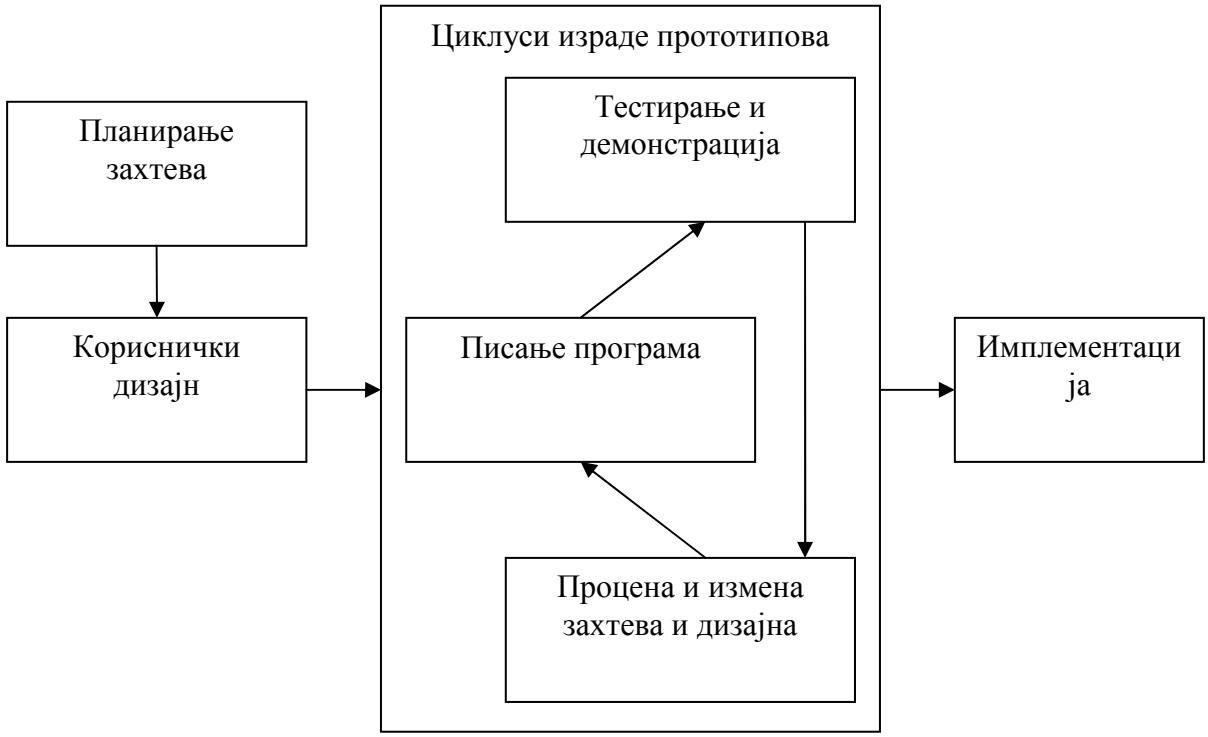
2.7.2.6 Чланови развојног тима

Када је у питању развојни тим, за брзи развој апликација су препоручени мали тимови, састављени од искусних, мотивисаних и флексибилних чланова, који могу да обављају различите улоге у тиму. Идеалан развојни тим има претходног искуства у брзом развоју апликација, изузетно познаје технологије, алате и софтверске компоненте са којима ради и финансијски је и временски дисциплинован.

2.7.3 Организација процеса развоја

Модел брзог развоја апликација се састоји од 4 фазе. Називи ових фаза су планирање захтева, кориснички дизајн, израда и имплементација.

По активностима које се у њима спроводе, фазе модела брзог развоја апликација, у извесној мери, имају своје еквиваленте у раније поменутим фазама процеса развоја софтвера. Тако, фаза планирања захтева одговара фази анализе и дефинисања захтева, фаза корисничког дизајна одговара фазама пројектовања система и пројектовања програма, фаза израде је циклична израда прототипова у којој се спроводе активности писања програма, тестирања и редизајнирања (поновна анализа и дефинисање захтева), а фаза имплементације одговара фази испоруке и одржавања. У литератури се оваква организација процеса развоја назива и еволутивна израда прототипа.



Слика 8 - Модел брзог развоја апликација

У фази планирања захтева, организују се састанци аналитичара и будућих корисника система, на којима се прави листа иницијалних захтева. Аналитичари дефинишу кључне функције у систему и уочавају ентитете значајне за те функције. Резултат ове фазе треба да буде листа ентитета и дијаграми акција који дефинишу односе између функција и података. Ова листа не мора бити, и обично није, комплетна и коначна за систем, али треба да у потпуности заокружи све кључне и већину осталих функционалности и буде довољна за прелазак у наредну фазу процеса. Аналитичари не би требало да представљају своја запажања у неструктурираном облику, већ треба да користе алате за моделирање захтева.

У фази корисничког дизајна, аналитичари и пројектанти заједнички раде на детаљнијем представљању захтева, праве прецизније дијаграме (концептулне, релационе, дијаграме акција), одређују који делови система су приоритетни, развијају планове тестирања, предлажу решења корисничког интерфејса и идентификују делове система који могу бити израђени помоћу већ постојећих софтверских компоненти. У овој фази се провизорно

планира и неколико наредних циклуса развоја, да би се омогућило једноставније мењање прототипа који настане у првом циклусу развоја.

Фаза израде се састоји од једног или више циклуса израде прототипова. Циклус се састоји од три подфазе – писање програма, тестирање и демонстрација и процена и измена захтева и дизајна. Програмери пишу програм по упутствима пројектаната. Употребом генератора кода, писање програма се вишеструко убрзава. Генератори кода могу претворити моделе података у потпуно функционалну базу података, генерисати читаве делове програма или корисничког интерфејса. Врло често генераторима кода се прави неко иницијално решење које се затим додатно прилагођава. Време је кључни елемент писања програма, и ако нека планирана функционалност није имплементирана, она се помера у следећи циклус израде. Након израде, прототип се тестира по плану развијеном у фази корисничког дизајна и демонстрира корисницима. Након демонстрације аналитичари, пројектанти и програмери заједнички раде на утврђивању захтева за следећи циклус израде и по потреби редизајнирају систем и програм.

Када се приближи време испоруке, развојни тим ће активније радити на изради техничке и корисничке документације и започети припреме за пуштање система у оперативни рад.

У фази имплементације систем се пушта у оперативни рад, обучавају се корисници, решавају се евентуални уочени проблеми у раду и прикупљају реакције корисника на систем.

2.7.4 Применљивост

Модел брзог развоја апликација је развијен као одговор на потребу да се готов систем испоручи веома брзо. Зато није погодан за све врсте пројеката. У табели су дати неки аспекти које треба анализирати да би се одредило да ли је систем погодан за модел брзог развоја апликација.

	Погодно за брз развој	Непогодно за брз развој
Делокруг рада система	Узан делокруг рада, у коме су циљеви и процедуре јасно дефинисани.	Широк делокруг рада, у коме су циљеви и процедуре нејасне и/или широко дефинисане.
Одлучивање	Одлуке о пројекту може донети мали број лако доступних људи.	Људи који доносе одлуке нису увек доступни или се у одлучивање о пројекту мора укључити велики број људи.
Поузданост	Поузданост није од критичног значаја.	Поузданост јесте од критичног значаја - системи за управљање критичним операцијама и системи за одржавање живота.
Перформансе	Нису неопходне оптималне перформансе.	Неопходне су оптималне перформансе система – симулације, рачунарске игре.
Подаци	Употреба постојећих података, главна функција система је анализа или извештавање о постојећим подацима.	Потребно је анализирати, пројектовати или прикупити велику количину сложених података.
Захтеване технологије	Захтевају се зреле, стабилне и проверене технологије.	Захтевају се технологије најновије генерације.
Архитектура система	Архитектура система је јасно дефинисана и кључне компоненте су тестиране и спремне.	Архитектура система није до краја разјашњена и планира се употреба компоненте или технологија које се користе први пут.
Софтверске компоненте	Већина функционалности се може постићи употребом постојећих софтверских компоненти.	Не постоје софтверске компоненте које покривају жељену функционалност.
Одржавање	Систем неће захтевати дуготрајно одржавање.	Систем ће захтевати дуготрајно одржавање.
Развојни тим	Развојни тим је мали, до 6 чланова, добро обучен, дисциплинован и мотивисан.	Развојни тим је велики, или постоји више тимова чији рад треба координирати.
Технички захтеви развоја	Технички захтеви развоја су разумни и могу бити задовољени постојећом опремом и технологијом. Оптерећеност развојних капацитета не треба да прелази 70%.	Технички захтеви развоја су блиски или прелазе ограничења постојеће опреме и технологије.

2.7.5 Предности и недостаци

Главне предности модела брзог развоја апликација су повећана брзина развоја, активна сарадња са корисником, велика флексибилност, снижени трошкови и смањен број дефеката у програму.

Развој се убрзава употребом алата за пројектовање уз помоћ рачунара, генератора кода, постојећих софтверских компоненти и наметањем временских ограничења.

Модел брзог развоја налаже да се корисник интензивно укључи у процес развоја. Корисник има потпун увид у процес и имплементиране функционалности, може рано да идентификује проблеме и предлаже решења. Има и надзорну улогу, па је смањена могућност да се развојни тим изгуби у захтевима и одлута од правог циља.

Флексибилност потиче од могућности да се током циклуса израде прототипа систем и програм више пута редизајнирају по најактуелнијим захтевима корисника и запажањима развојног тима.

Развојни тим се ослања на алате, па за краће време обавља своје задатке. Имплементација функционалности употребом постојећих софтверских компоненти и генератора кода може бити јефтинија од развоја тих функционалности од нуле.

Конверзија модела у изворни код и генератори кода у извесној мери елиминишу потребу за ручним писањем кода, умањујући могућности појаве дефеката у програму.

Главни недостаци модела брзог развоја апликација су смањена скалабилност, потенцијално смањење функционалности, теже мерење прогреса, мања ефикасност система и проблеми који потичу од употребе софтверских компоненти.

Пошто се модел фокусира на израду прототипа који се циклично допуњава до пуне функционалности, испорученом решењу може недостајати скалабилност коју би имао да је систем рађен у једном покушају и да су сви његови захтеви унапред разматрани. Овај проблем може да се превазиђе уз употребу квалитетног генератора апликација, који ће у старту обезбедити добро пројектовану, ефикасну и скалабилну апликацију.

Услед временског ограничавања приоритет постаје време испоруке, па се све недовршене функционалности премештају у наредну верзију, што може да узрокује смањење броја функционалности у верзији за испоруку.

Модел не обезбеђује јасну сигнализацију када је систем готов. Развој се завршава када се развојни тим и клијент сложе да је прототип довољно добар. Како се план развоја формира током циклуса израде прототипа, по тренутно актуелним захтевима, није увек једноставно закључити докле је укупан развој одмакао.

Изворни код који генеришу алати за конверзију модела и генератори кода не мора увек бити и најефикаснији. Може бити превише апстрактан, мање поуздан или ниских перформанси. Најефикаснији код ипак пише сам човек.

Употребом софтверских компоненти, развојни тим у извесној мери губи контролу над производом, јер није у могућности да имплементира све функционалности које жели, нити на начин на који жели. Систем постаје зависан од производа других произвођача, њихових стратегија развоја и финансијске стабилности, па је дугорочно одржавање неизвесно.

3 Пројекат и израда система за учење на даљину

3.1 Планирање захтева

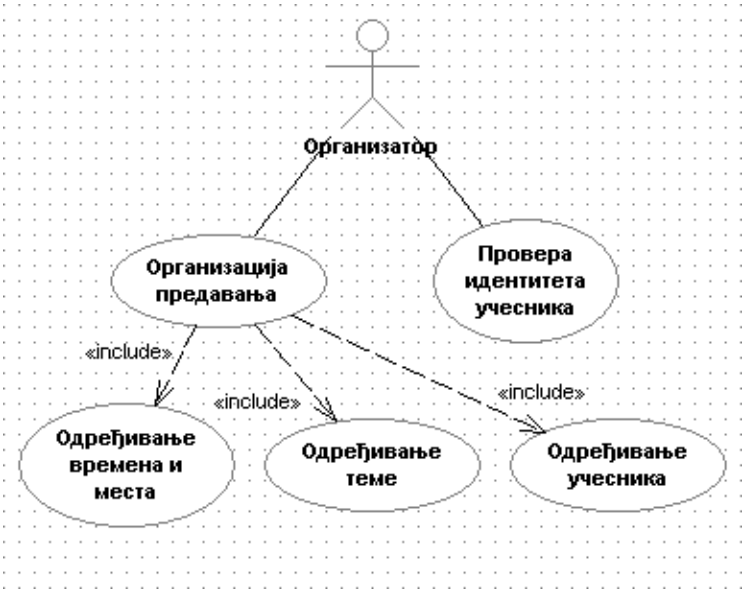
Планирање захтева започињемо анализом фотографије учионице у којој је предавање у току и наших постојећих знања о предавањима, уочавајући важне ентитете и односе у учионици и изван ње, у мери која нам је потребна за пројектовање система.

Слика 9 - Учионица током предавања

Најуочљивији ентитети су непосредни учесници предавања - предавач и слушаоци. Они учествују у предавању по некој организацији. Организација је одредила и време и место одржавања предавања. Организација може налагати да учесници морају имати основ по коме присуствују предавању (предавач мора бити неко кога акредитује организатор, слушалац мора уплатити новчани износ), па у ту сврху проверава идентитет сваког учесника. Предавање се не може одржати ако нема предавача. Предавач је један, слушалаца може бити више. Између учесника је успостављен комуникациони канал помоћу кога комуницирају у оба смера, предавач ка слушаоцима и слушаоци ка предавачу. У ситуацији на фотографији, комуникациони канал је успостављен присуством на истом месту у исто време. Предавање подразумева усмено излагање. У једном временском тренутку само један учесник предавања говори, а остали слушају. Пажња учесника је усмерена ка учеснику који тренутно говори. Слушаоци могу да постављају питања предавачу. Комуникација између слушалаца, уколико у њу није укључен предавач или није у директној вези са предавањем, је од секундарног значаја и може се сматрати непожељном. Током предавања, користе се помагала попут табле или слике са пројектора које слушаоци у потпуности виде. По табли може да се пише, а помоћу пројектора се приказује наставни материјал. Наставни материјал може бити у форми слајда, текста, слике, изворног кода, аудио или видео снимка и слично. Било који учесник може

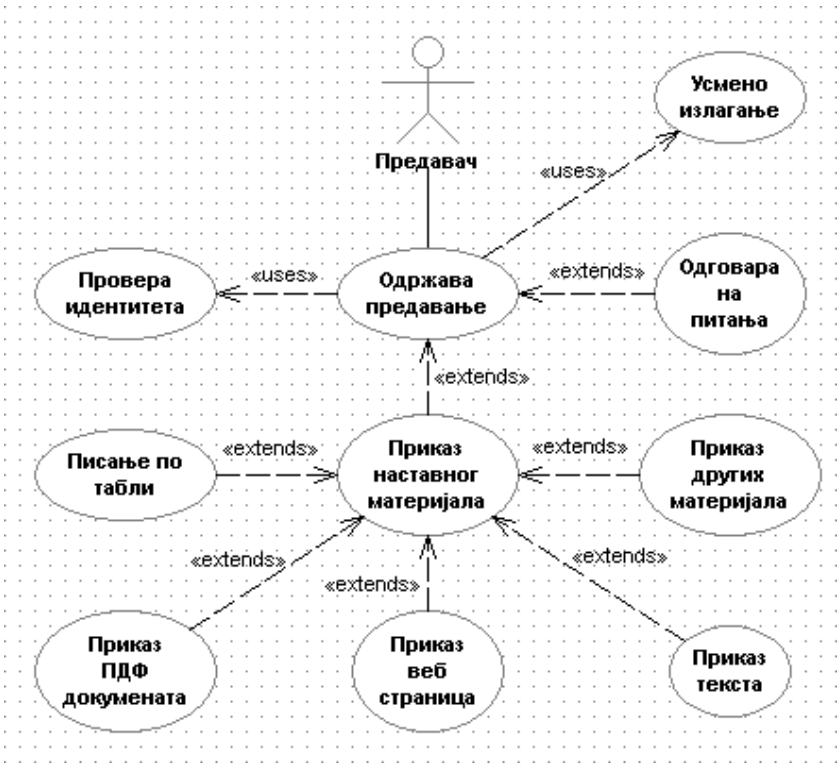
напустити предавање у било ком тренутку. Ако предавач напусти предавање, предавање је завршено.

Из ове опште анализе ћемо моделовати учеснике, њихове активности и редослед тих активности.



Слика 10 - Дијаграм активности организатора предавања

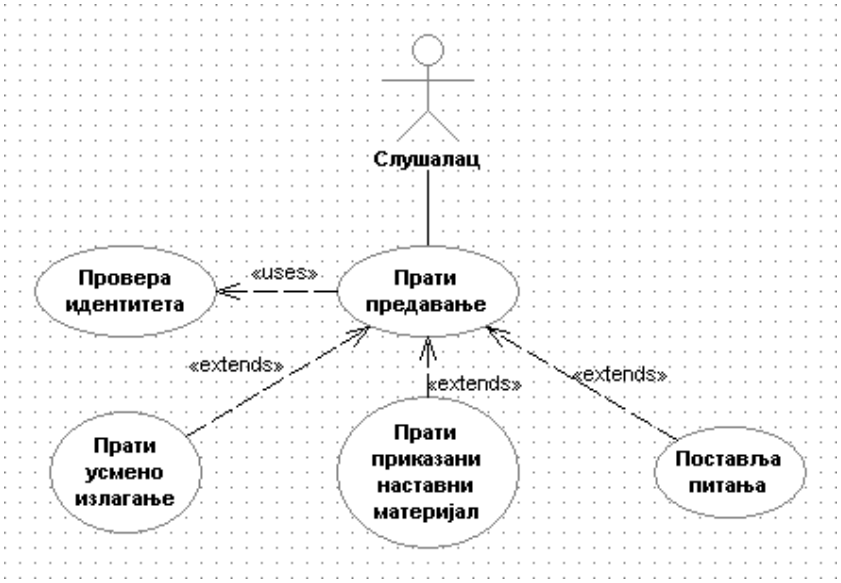
Из дијаграма активности организатора се види да он организује предавање и проверава идентитет учесника. Организација предавања је активност која се састоји од одређивања времена и места предавања и одређивања учесника предавања. Организатор је само један.



Слика 11 - Дијаграм активности предавача

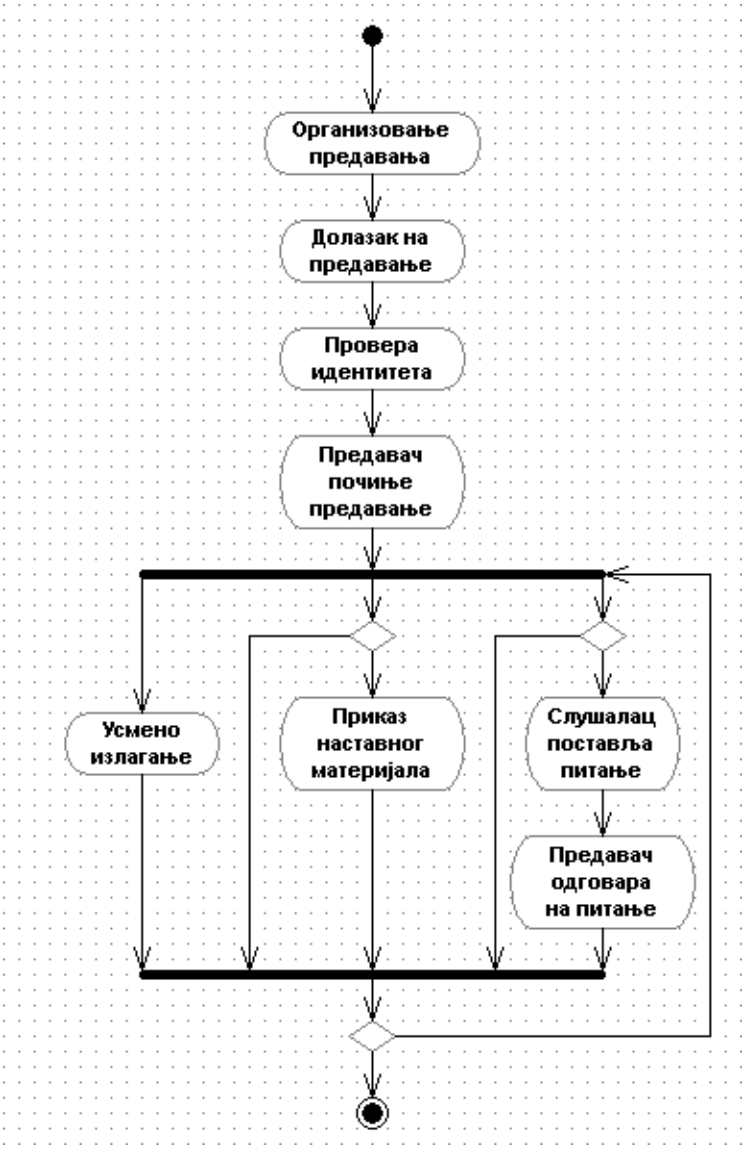
Из дијаграма активности предавача се види да он може да одржава предавање, што укључује проверу његовог идентитета и усмено излагање. Предавач ће одговарати на постављена питања, а предавање се може допунити приказом наставног материјала, што

се реализује писањем по табли, или приказивањем ПДФ докумената, веб страница, текста или других погодних материјала. Предавач је само један.



Слика 12 - Дијаграм активности слушаоца

Из дијаграма активности слушаоца предавања се види да он може да прати предавање, што укључује проверу његовог идентитета. Током предавања, слушалац прати усмено излагање предавача и приказани наставни материјал, а може и да поставља питања предавачу. Слушалаца може бити више.



Слика 13 - Дијаграм тока предавања

Из дијаграма активности се види како тече једно предавање. Први корак је организовање предавања, чиме се одређују време и место одржавања, учесници и тема предавања. Учесници ће у договорено време доћи на договорено место. У случају који разматрамо, организација захтева основу присуства, па се врши и провера идентитета, након чега предавач започиње предавање. Предавање се одвија у циклусима усмених излагања, током кога се може приказивати наставни материјал или одговарати на постављена питања. У једном тренутку овај циклус ће бити прекинут (предавач је завршио своје излагање или је истекло време предвиђено за предавање) и предавање се завршава.

Нефункционалних захтева нема.

Као пројектни захтев постављамо ограничење да се програм изради као десктоп апликација за оперативни систем *Мајкрософт Виндоуз*.

Као процесне захтеве постављамо употребу модела брзог развоја апликација и ограничење трајања једног циклуса израде прототипа на 20 радних дана, подељено по фазама у циклусу на 14 + 3 + 3 дана.

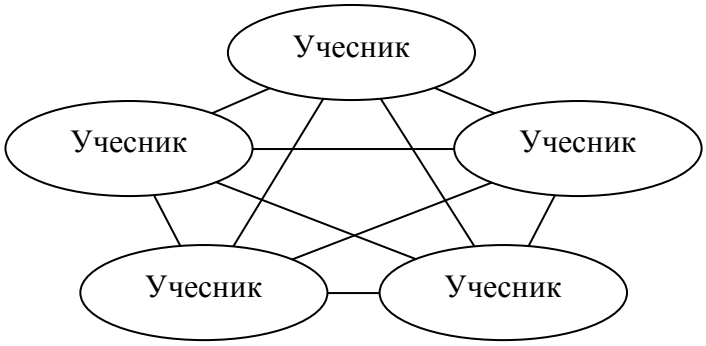
3.2 Кориснички дизајн

Да бисмо реализовали процес учења на даљину, потребно је да све ентитете, односе међу њима и њихове активности које смо уочили у конвенционалној учионици, што приближније реализујемо у оквиру рачунарског окружења.

3.2.1 Архитектура система

Прва одлука у пројектовању система се тиче архитектуре решења. Јасно је да ће комуникациони канал између учесника бити рачунарска мрежа, тј. Интернет. У нашем случају, комуникациони канал подразумева да се сви људи међусобно чују и виде, па се намеће потреба да у систему користимо пренос аудио и видео података у реалном времену. Управо ова ставка, комуникација између учесника, је кључна за одлуку о архитектури решења. Под претпоставком да сваки корисник шаље исту количину аудио и видео података, анализираћемо два могућа решења.

Ако бисмо комуникацију између учесника реализовали као децентрализован систем, тако да сваки учесник сваком другом учеснику директно шаље свој, и директно прима све друге аудио и видео податке, за једног учесника бисмо имали линеарни раст и одлазног и долазног саобраћаја са порастом броја учесника.



Слика 14 - Децентрализовани систем комуникације

У централизованом систему, сваки учесник шаље своје податке посреднику, а посредник их прослеђује осталим учесницима. У овој реализацији, за једног учесника долазни саобраћај и даље има линеарни раст са порастом броја учесника, али је одлазни саобраћај константан, тј. највећи део оптерећења одлазног саобраћаја је премештен на посредника.

Слика 15 - Централизовани систем комуникације

Друга важна ставка за избор архитектуре је веза са Интернетом преко које ће учесници учествовати на предавању. У случају децентрализованог система, однос долазног и одлазног саобраћаја је 1:1. У случају централизованог система, однос долазног и одлазног саобраћаја је (број учесника – 1):1. Учесници ће учествовати на предавању углавном преко кућне везе са Интернетом, а оне су данас углавном асиметричне, тј. имају већу долазну него одлазну брзину. Централизовани систем је прилагођен оваквом мрежном окружењу.

Трећа важна ставка за избор архитектуре је потреба за провером идентитета учесника, што је по својој природи централизован процес.

Имајући све ово у виду, јасно је да ћемо изабрати централизовану клијент-сервер архитектуру. Централизовани систем јесте скупљи, јер захтева инсталацију и одржавање посредника на брзој вези са Интернетом, али може да ради под много већим оптерећењем од децентрализованог система.

3.2.2 Функционалности сервера

Серверски део система треба да покрије функционалности организатора, тј. организовање предавања и проверу идентитета учесника, и да посредује у комуникацији између учесника.

За проверу идентитета учесника ћемо применити стандардни приступ – идентитет ћемо проверавати одмах по повезивању на сервер и учесник неће моћи да користи систем док се његов идентитет не провери и потврди.

За организацију предавања ћемо применити концепт тзв. соба за ћаскање (chatroom), који се често среће у програмима за размену текстуалних порука преко Интернета (на пример ИРЦ). Пошто је присуство предавача обавезно за предавање, предавање ће почињати тако што ће предавач направити групу којој слушаоци могу да се придруже. Сви учесници у једној групи учествују на истом предавању. Можемо омогућити истовремено постојање више оваквих група, чиме омогућавамо да се истовремено одвија више предавања. У том случају, групе морају бити међусобно изоловане и искључиве на нивоу корисника, тј. један учесник у једном тренутку може бити само у једној групи. Аналогија између оваквих група учесника и учионица је сасвим очигледна.

Сервер ће обезбедити командни протокол за проверу идентитета и управљање групама учесника.

Комуникација на предавању подразумева да се учесници међусобно чују и виде и да могу да прате наставни материјал који се приказује. Сви учесници ће серверу слати своје аудио и видео податке, а предавач ће додатно слати и наставни материјал. Када прими податке сервер ће их проследити свим другим учесницима у групи.

3.2.3 Функционалности клијента

Клијентски део система треба да покрије функционалности предавача и слушаоца. Заједничке функционалности за предавача и слушаоца су провера идентитета и размена аудио и видео података између учесника. Додатно, предавачу треба омогућити да направи групу учесника и представља наставни материјал, а слушаоцу да се придружи групи учесника и прати наставни материјал.

Функционалности провере идентитета и управљање групама учесника ћемо обезбедити имплементацијом клијентске стране командног протокола сервера.

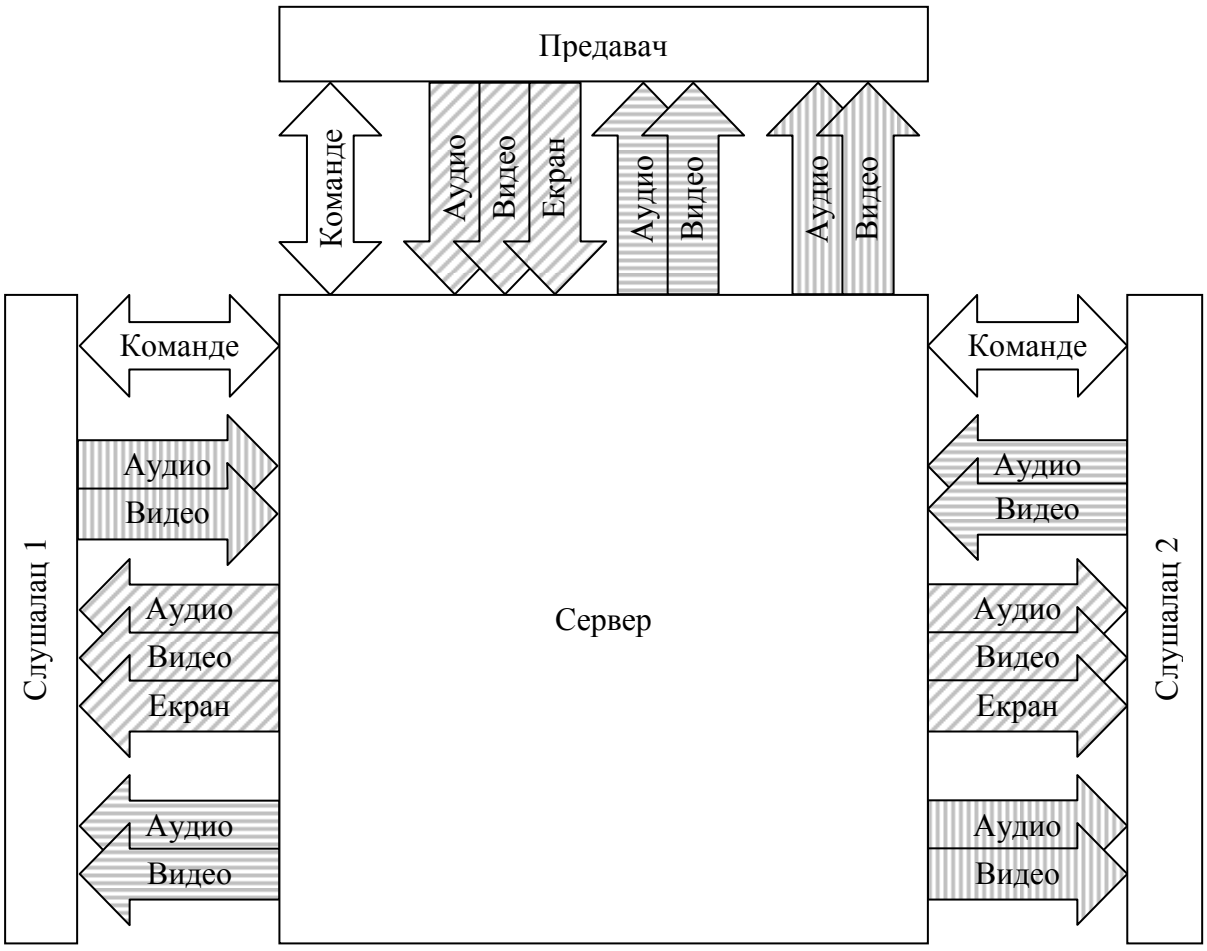
Аудио и видео податке ћемо прикупљати помоћу микрофона и камере, и преносити их у реалном времену.

Како наставни материјал може бити веома различите природе, најбољи начин да омогућимо његово представљање је да снимак екрана предавача такође шаљемо у реалном времену. Представљање наставног материјала захтева и таблу за цртање и писање, па ћемо омогућити да предавач користи графичку таблу са оловком као таблу за цртање и писање.

3.2.4 Мрежни саобраћај

Из свега до сада наведеног, можемо дати груби приказ мрежног саобраћаја током трајања предавања.

На примеру предавања са три учесника, видимо да предавач шаље своје аудио и видео податке и снимак екрана, а прима аудио и видео податке од слушалаца. Први слушалац шаље своје аудио и видео податке, а прима податке од предавача и другог слушаоца. Други слушалац шаље своје аудио и видео податке, а прима податке од предавача и првог слушаоца. Видимо и да сваки учесник има и командни интерфејс ка серверу.



Слика 16 - Мрежни саобраћај

3.2.5 Развојни алати

Развојно окружење за имплементацију система је *Ембаркадеро РАД Студио* 2009, а програмски језик је *Објектни Паскал*.

> *Ембаркадеро РАД Студио* је наследник развојног окружења *Борланд Дивелопер Студио* (Borland Developer Studio), које је настало обједињавањем два водећа развојна окружења компаније *Борланд* (Borland) – *Борланд Делфи* (Borland Delphi) и *Борланд Ц++ Билдер* (Borland C++ Builder). Након промене пословне оријентације, *Борланд* је првобитно издвојио производњу развојних окружења у посебну компанију *Кодгир* (CodeGear), а затим и потпуно напустио производњу развојних окружења и *Кодгир* продао компанији *Ембаркадеро*. Данас, уз *Вижуал Ц++* (Visual C++), *РАД студио* представља стандард за развој тзв. *нејтив* програма (native, тј. програма који се преводе у машинске инструкције) за *Мајкрософт Виндоуз* платформу. У овом домену, *РАД студио* је и једино окружење индустријског квалитета прилагођено брзом развоју апликација.

За моделирање, визуелизацију и генерисање изворног кода вишег нивоа (класе), ћемо користити и *Модел Мејкер*. У имплементацији ћемо користити и *Модел Мејкер Код Иксплорер* (Modelmaker Code Explorer), чије су главне функционалности структурирана навигација кроз изворни код, рефакторисање и генерисање изворног кода нижег нивоа (методе). Оба алата су интегрисана у *РАД Студио*.

3.2.6 Софтверске компоненте и библиотеке

Кориснички интерфејс може бити израђен помоћу *библиотеке визуелних компоненти* (Visual Component Library, VCL). Компоненте за имплементацију графичког корисничког интерфејса из ове библиотеке су углавном омотачи (wrapper) стандардних графичких елемената корисничког интерфејса оперативног система *Мајкрософт Виндоуз*. Библиотека визуелних компоненти је стандардни део *РАД Студија*.

За реализацију табле за цртање и писање можемо користити *Актив Икс* (ActiveX) компоненте *Инк Пикчр* (Ink Picture), из пакета *Мајкрософт Таблет Пи Си Тајп Лајбрари* (Microsoft Tablet PC Type Library) и *Инк Едит* (Ink Edit), из пакета *Мајкрософт Виндоуз Икс Пе Таблет Пи Си Едишн Рекогнајзер Пек* (Microsoft Windows XP Tablet PC Edition Recognizer Pack). Прва компонента имплементира напредне функционалности цртања и писања помоћу графичке табле са оловком. Друга компонента имплементира препознавање рукописа. Пакети су слободно доступни за преузимање са веб странице произвођача.

За приказ ПДФ докумената можемо користити *Актив Икс* компоненту *Акро ПДФ* (AcroPDF), из пакета *Адоби Акробат Браузер Тајп Лајбрари* (Adobe Acrobat Browser Type Library). Компонента имплементира функционалности програма *Адоби Акробат Ридер* (Adobe Acrobat Reader), који је стандардни читач за ПДФ документе. Компонента је слободно доступна и налази се у инсталационом програму за *Адоби Акробат Ридер*.

За приказ веб страница можемо користити компоненту *Веб Браузер* (WebBrowser) из *библиотеке визуелних компоненти*, која је омотач *Актив Икс* компоненте *Веб Браузер* из пакета *Мајкрософт Интернет Контролс* (Microsoft Internet Controls). Компонента имплементира функционалности веб читача *Мајкрософт Интернет Иксплорер* (Microsoft Internet Explorer). *Мајкрософт Интернет Контролс* је део инсталације оперативног система *Мајкрософт Виндоуз*.

За приказ текстуалних датотека можемо користити компоненту *Адвансд Мемо* (Advanced Memo) из пакета *ТМС Компонент Пек* (TMS Component Pack). Компонента имплементира функционалности текстуалног едитора са бојењем синтаксе. *ТМС Компонент Пек* је комерцијална развојна библиотека за *РАД Студио*.

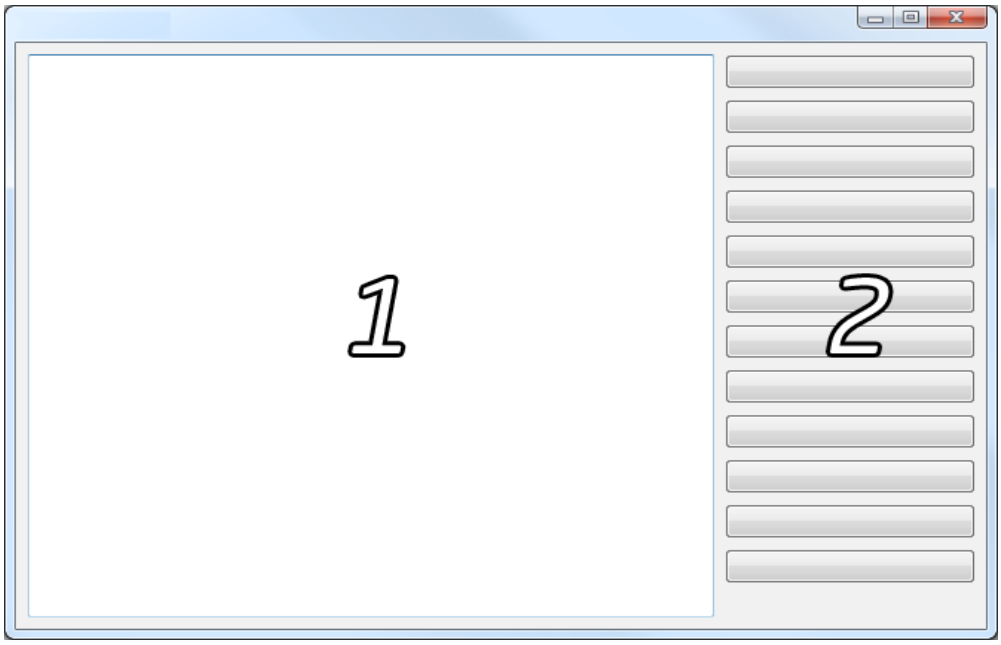
За прикупљање и обраду аудио и видео података са микрофона и камере за аудио и видео стриминг можемо користити компоненте из пакета *Митов Аудио Лаб* (Mitov Audio Lab), *Митов Видео Лаб* (Mitov Video Lab) и *Митов Сигнал Лаб* (Mitov Signal Lab). Компоненте имплементирају функционалности прикупљања и обраде аудио и видео сигнала, помоћу више *Мајкрософт Виндоуз* технологија. *Митов Аудио Лаб, Митов Видео Лаб* и *Митов Сигнал Лаб* су комерцијалне развојне библиотеке за *РАД Студио*, слободно доступне за преузимање са веб странице произвођача и бесплатне за некомерцијалну употребу.

За реализацију мрежне комуникације можемо користити компоненте из пакета *Инди Сокетс* (Indy.Sockets). Компоненте имплементирају бројне стандардне и нестандардне протоколе апликативног нивоа у оквирима ТЦП и УДП протокола. *Инди Сокетс* је развојна библиотека отвореног кода и стандардни део *РАД Студија*.

Приметићемо да је већина функционалности које нам сугерише досадашња анализа покривена функционалностима постојећих софтверских компоненти и библиотека. У даљем пројектовању ћемо се ослонити на ове компоненте, тј. сматраћемо да је њихова употреба у имплементацији система решена ствар, и нећемо се бавити анализом или пројектовањем функционалности које оне имплементирају (примера ради, нећемо се бавити анализом и пројектовањем табле за цртање и писање). Сматраћемо да смо наведене функционалности обезбедили, а евентуалним недостацима ћемо се бавити у наредним циклусима израде.

3.2.7 Графички кориснички интерфејс сервера

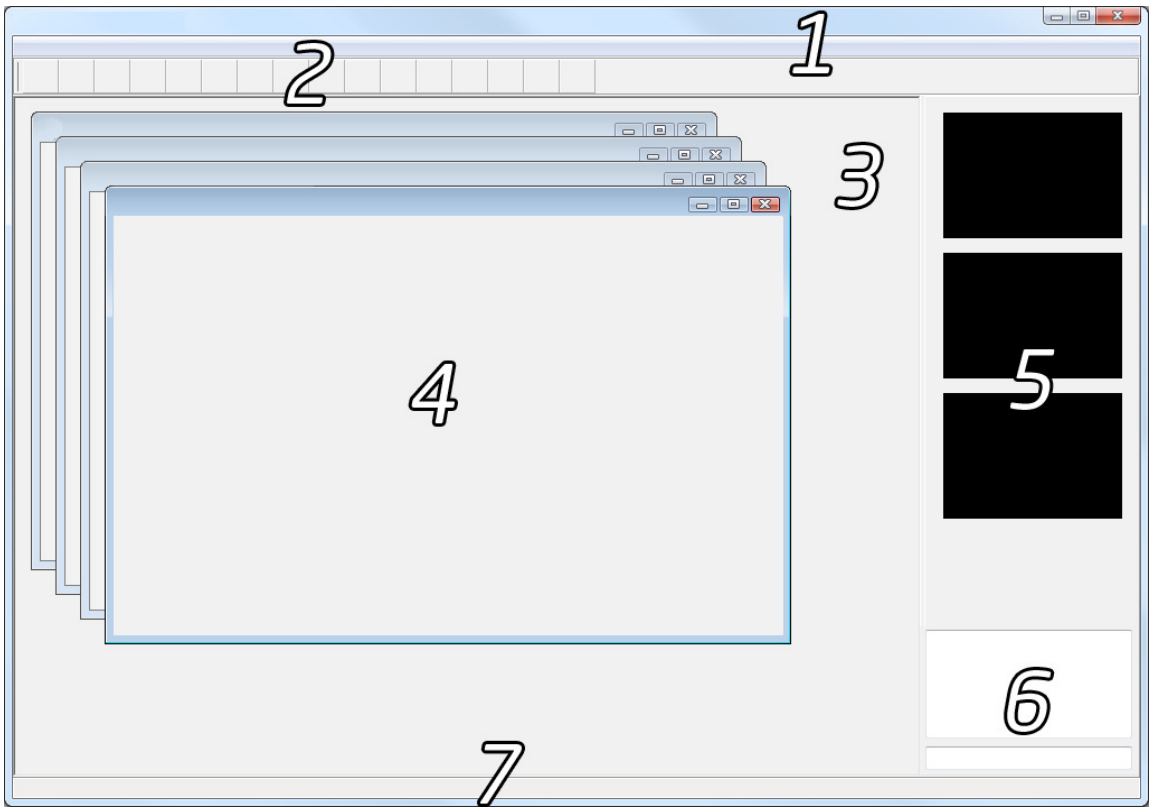
У финалној верзији, сервер треба да буде израђен као сервисна апликација, тако да му није потребан кориснички интерфејс. Ипак, у првом циклусу ћемо га израдити као корисничку апликацију, што ће нам олакшати проналажење и исправљање проблема у систему. Кориснички интерфејс треба да садржи простор за испис дијагностичких порука у реалном времену (1) и контролне дугмиће (2).



Слика 17 - Предлог графичког корисничког интерфејса сервера

3.2.8 Графички кориснички интерфејс клијента

За основу корисничког интерфејса клијента бирамо тзв. *МДИ интерфејс* (Multi Document Interface), уз неколико додатака: мени (1), палета алатки (2), простор за подпрозоре (3), подпрозори за приказ наставног материјала (4), простор за приказ видео снимака учесника (5), простор за размену текстуалних порука (6) и статусна линија (7).



Слика 18 - Предлог графичког корисничког интерфејса клијента

Подпрозори за приказ наставног материјала ће имати сопствене палете алатки са командама карактеристичним за своју функционалност (на пример, подпрозор који ће приказивати веб читач ће имати команде за навигацију). Тачне функционалности ћемо утврдити када одредимо тачан начин имплементације функционалности подпрозора за приказ наставног материјала.

3.2.9 Командни протокол сервера

Рекли смо да ће сервер обезбедити командни протокол за проверу идентитета и управљање групама. Чак и сасвим површна анализа потребних команди нам указује да је извршавање неких команди логично само у одговарајућем контексту. На пример, планирали смо да учесник неће моћи да користи систем пре провере идентитета, тако да команда за проверу идентитета мора вратити позитиван резултат пре издавања било које друге команде. Или, јасно је да учесник не може напустити предавање ако му се претходно није прикључио.

Овде дајемо предлог скупа команди, стања и коначног аутомата који одређује контекст извршавања тих команди. Коначни аутомат ћемо користити у имплементацији командног протокола сервера, али и у имплементацији клијента, за прилагођавање елемената корисничког интерфејса, на пример, да искључимо елемент корисничког интерфејса који се користи за издавање команде чије издавање нема смисла у тренутном контексту.

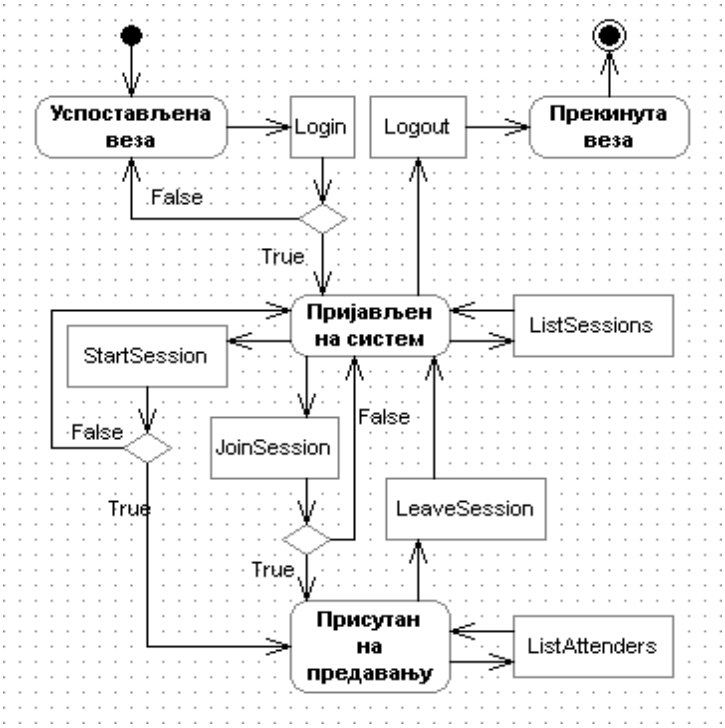
У следећој табели је дат преглед предложених команди и њихов опис.

Команда: **`Login username password`** Идентификација учесника. Параметри су корисничко име и лозинка. Резултат је позитиван или негативан, у зависности од тога да ли је пар корисничко име и лозинка пронађен у листи могућих учесника.
Команда: **`Logout`** Одјављивање са система и прекид успостављене везе. Нема параметара, ни повратну вредност.
Команда: **`ListSessions`** Приказује листе тренутно активних предавања. Нема параметара. Резултат је листа имена тренутно активних предавања.
Команда: **`StartSession unique_group_name`** Прави групу учесника под задатим именом, и поставља учесника у позицију предавача. Параметар је јединствени назив предавања. Резултат је позитиван ако група са задатим именом не постоји, у супротном је негативан.
Команда: **`JoinSession existing_group_name`** Придружује учесника постојећој групи и поставља га у позицију слушаоца. Параметар је назив групе учесника. Резултат је позитиван ако група са задатим именом постоји, у супротном је негативан.
Команда: **`ListAttenders`** Приказује листу учесника предавања. Нема параметара. Резултат је листа имена свих учесника предавања.
Команда: **`LeaveSession`** Напуштање предавања. Нема параметара, ни повратну вредност.

У следећој табели је дат преглед предложених стања и њихов опис.

Стање: Успостављена веза Клијент се повезао са сервером.
Стање: Пријављен на систем Клијент је послао корисничко име и лозинку и његов идентитет је потврђен.
Стање: Присутан на предавању Клијент је постао учесник на предавању, тј. покренуо је ново предавање или се прикључио неком постојећем предавању.
Стање: Прекинута веза Клијент се одјавио из система, тј. престаје да користи систем.

На следећем дијаграму је приказан коначни аутомат дефинисан за предложене команде и стања.



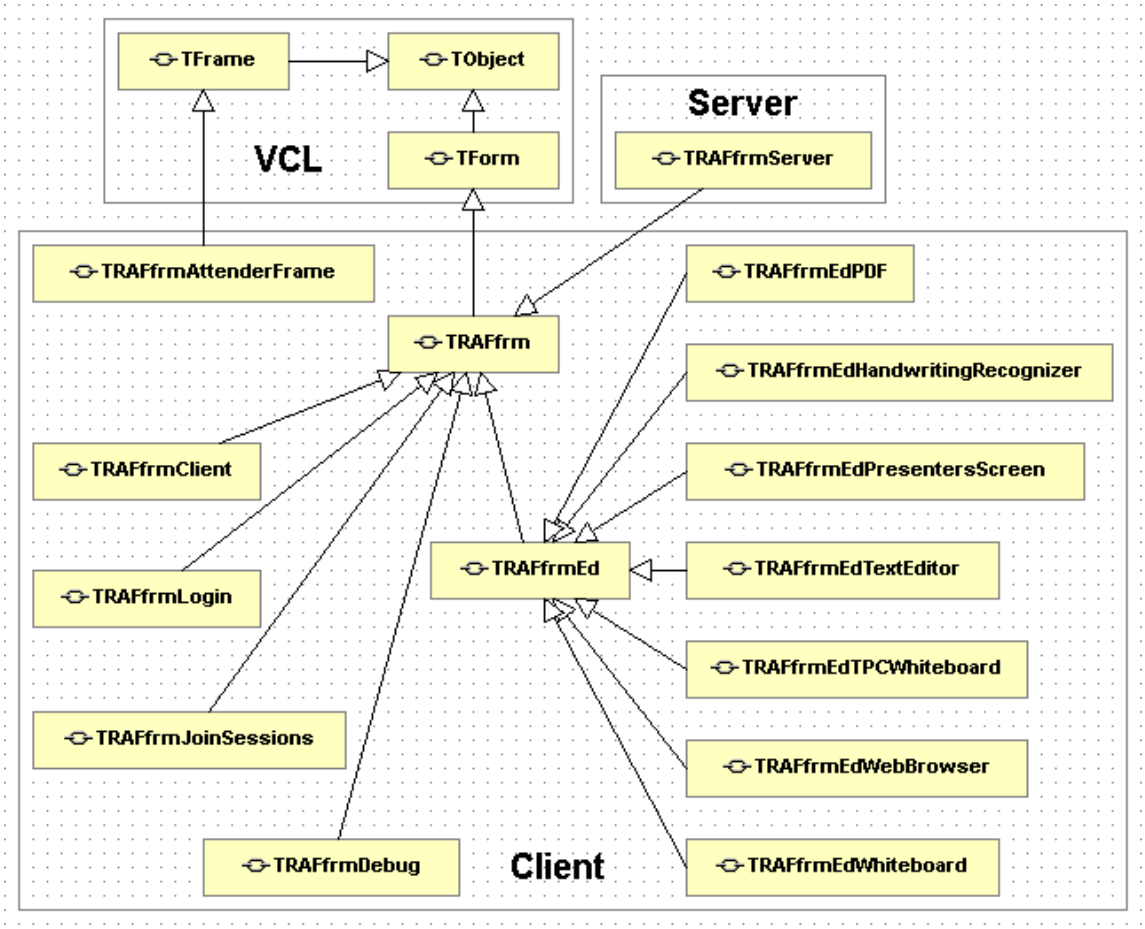
Слика 19 - Коначни аутомат за предложене команде и стања

Почетно стање је тренутак успостављања везе са сервером. У овом стању корисник може издавати само команду **`Login`**, којом се идентификује систему. Ако команда **`Login`** врати позитиван резултат, учесник прелази у стање пријављен на систем, и може издавати друге команде. Командом **`Logout`** се може одјавити из система, чиме се и прекида успостављена веза. Командом **`ListSessions`** може добити листу текућих предавања. Командом **`StartSession`** може покренути ново предавање, чиме учесник постаје предавач на предавању. Командом **`JoinSession`** се може придружити текућем предавању, чиме учесник постаје слушалац на предавању. Команде **`StartSession`** и **`JoinSession`** мењају стање у присутан на предавању, па учесник може издати и команде **`ListAttenders`**, којом добија листу других учесника на предавању, и **`LeaveSession`** којом напушта предавање и враћа се у стање пријављен на систем.

3.2.10 Класни дијаграм корисничког интерфејса

У овом делу дајемо преглед класа и хијерархије класа које ћемо користити за израду првог прототипа. Како ћемо већину функционалности имеплементирати помоћу софтверских компоненти и библиотека, радно окружење и генератори кода ће током писања кода аутоматски дефинисати највећи део потребних чланова и метода класа и извршити повезивање догађаја и метода, тако да детаљно моделирање сваке класе нема пуно смисла. Тамо где буде потребно, разрадићемо детаљнији модел.

Следећи дијаграм је хијерархија класа графичког корисничког интерфејса. Видимо три издвојене целине. Једна целина је библиотека визуелних компоненти, приказана само ради потпуности хијерархије. Друга целина обухвата класе у корисничком интерфејсу сервера, а трећа класе у оквиру корисничког интерфејса клијента. Описаћемо најважније.



Слика 20 - Хијерархија класа графичког корисничког интерфејса

Класа **`TObject`** је базна класа за све класе у библиотеци визуелних компоненти.

Класа **`TForm`** имплементира прозор, и она је основа графичког корисничког интерфејса.

Класа **`TFrame`** је помоћна класа за израду напредних елемената графичког корисничког интерфејса. TFrame може да обједини више једноставних елемената корисничког интерфејса и са њима ради као једном целином.

Класа **`TRAFfrm`** је базна класа за све прозоре у систему, без обзира да ли се ради о главном прозору, подпорзорима или дијалозима. Имплементираће функционалности позиционирања прозора и локализације.

Класа **`TRAFfrmServer`** ће имплементирати главни прозор корисничког интерфејса сервера.

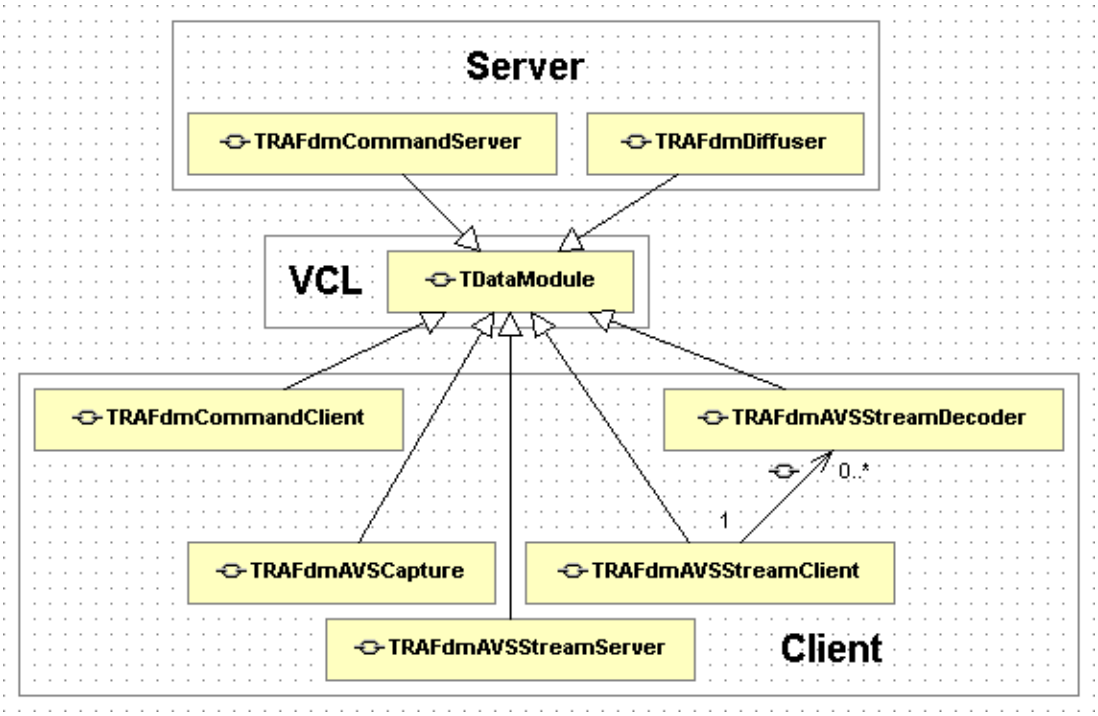
Класа **`TRAFfrmEd`** је базна класа за све подпрозоре за приказ наставног материјала. Сваки наследник ове класе ће имплементирати функционалност за приказивање неке врсте наставног материјала. Тако ће класа типа **`TRAFfrmEdPDF`** имплементирати читач ПДФ докумената, класа **`TRAFfrmEdWebBrowser`** ће имплементирати веб читач, а класа **`TRAFfrmEdWhiteboard`** ће имплементирати таблу за цртање и писање.

Класа **`TRAFfrmClient`** ће имплементирати главни прозор корисничког интерфејса клијента.

Класе **TRAFfrmLogin**, **TRAFfrmJoinSession**, **TRAFfrmDebug** ће имплементирати помоћне дијалоге за пријављивање на систем, придруживање групи учесника и приказ дијагностичких порука.

3.2.11 Класни дијаграм подсистема за пренос података

Следећи дијаграм је хијерархија класа подсистема за размену података преко рачунарске мреже. И овде видимо три целине које се односе на библиотеку визуелних компоненти, сервер и клијент.



Слика 21 - Хијерархија класа подсистема за размену података преко рачунарске мреже

Класа **TDataModule** је базна класа за све класе подсистема за размену података.

Класа **TRAFdmCommandServer** ће имплементирати серверску страну командног протокола.

Класа **TRAFdmDiffuser** ће имплементирати пријем и слање аудио и видео података учесника.

Класа **TRAFdmCommandClient** ће имплементирати клијентску страну командног протокола.

Класа **TRAFdmAVSCapture** ће имплементирати прикупљање аудио и видео података и снимка екрана.

Класа **TRAFdmAVSStreamServer** ће имплементирати кодирање и компресовање аудио и видео података и снимка екрана и слање ових података посредничком серверу.

Класа **TRAFdmAVSStreamClient** ће имплементирати пријем аудио и видео података и снимка екрана од посредничког сервера.

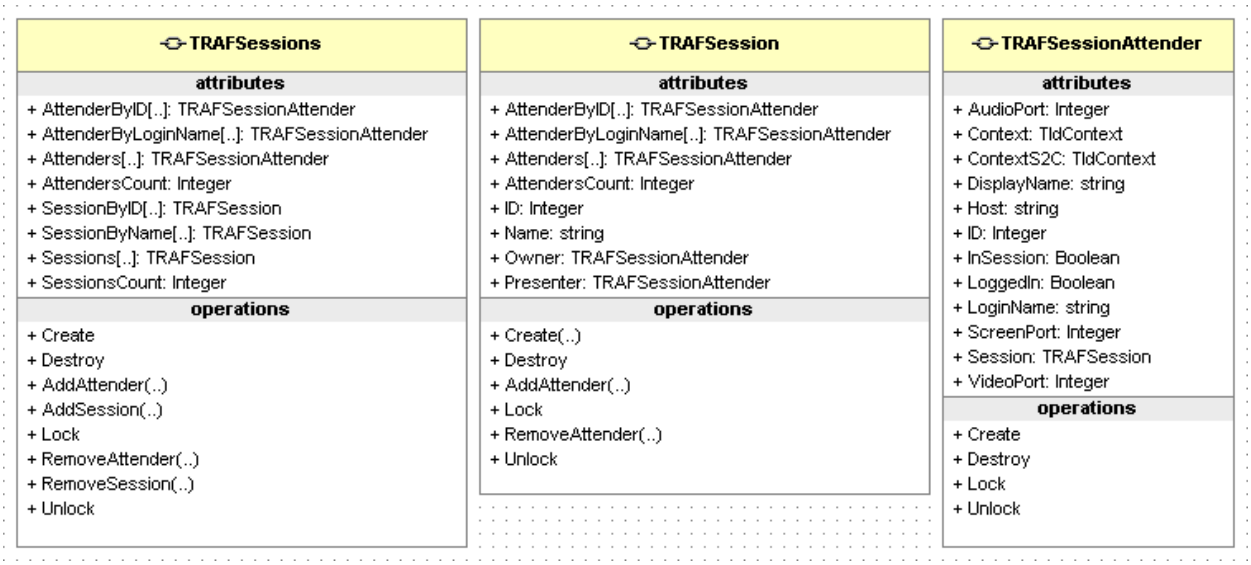
Класа **TRAFdmAVSStreamDecoder** ће имплементирати декомпресовање и декодирање аудио и видео података и снимка екрана. На дијаграму се види кардиналност 1:0..* између

TRAFdmAVSStreamClient и **TRAFdmAVSStreamDecoder**. **TRAFdmAVSStreamClient** ће примати податке од свих учесника, а за сваког учесника посебно ће направити једну инстанцу **TRAFdmAVSStreamDecoder**.

Објаснићемо укратко зашто је класа **TDataModule** узета као базна класа за све класе подсистема за размену података. Највећи број класа из развојних библиотека за *РАД Студио* су имплементиране као тзв. компоненте, тј. као наследнице класе **TComponent**. Компоненте се деле на визуелне и невизуелне. Визуелне компоненте током извршавања програма имају графички приказ и то су прозори, менији, дугмићи и слично. Невизуелне компоненте током извршавања програма немају графички приказ. Невизуелна компонента може имплементирати комуникацију са базом података, временски окидач, парсер и слично. Међутим, у развојном окружењу, током писања програма, све компоненте имају графички приказ и њиховим својствима се може веома једноставно манипулисати без писања изворног кода. Основна намена класе **TDataModule** је да централизује невизуелне компоненте и да омогући једноставну манипулацију њима. У нашем случају, како ћемо у овом делу програма користити велики број невизуелних компоненте из пакета *Митов Аудио Лаб, Митов Видео Лаб, Митов Сигнал Лаб* и *Инди Сокетс*, избор **TDataModule** за базну класу је логичан.

3.2.12 Класни дијаграм подсистема за управљање групама корисника

На следећем класном дијаграму је приказана детаљна структура неколико класа које ћемо користити за управљање групама корисника. Ове класе ће бити имплементиране у серверу, а њима ће се манипулисати преко командног интерфејса. Њихова функција је да чувају податке о учесницима у постојећим групама учесника. **TRAFdmDiffuser** ће користити ове класе за прослеђивање података које прими.



Слика 22 - Дијаграм класа за управљање групама корисника

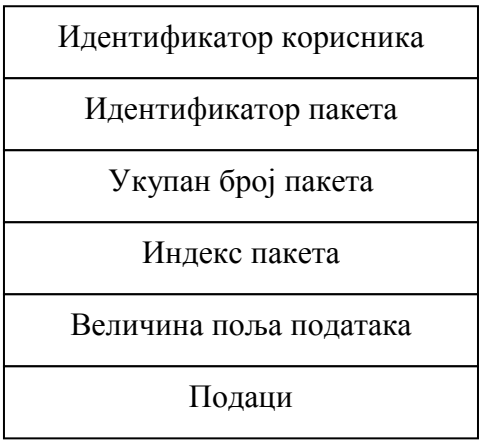
Класа **TRAFSessions** ће имплементирати листу свих група учесника које су направљене на серверу. Садржаће методе и својства неопходне за добијање информација о групама учесника или учесницима по различитим параметрима.

Класа **`TRAFSession`** ће имплементирати групу учесника, тј. листу учесника у једној групи. Садржаће методе и својства неопходне за добијање информација о групи и учесницима по различитим параметрима.

Класа **`TRAFSessionAttender`** ће имплементирати учесника предавања. Садржаће својства за чување информација о учеснику, параметрима везе и статусу учесника у оквиру постојећих група.

3.2.13 Структура мрежног пакета

Следећа слика приказује структуру мрежног пакета којим ћемо преносити аудио и видео податке и снимак екрана.



Слика 23 - Структура мрежног пакета

Идентификатор корисника је јединствени идентификациони број на нивоу сервера који се користи за идентификацију корисника који је послао пакет.

Идентификатор пакета је јединствени идентификациони број на нивоу сервера који се користи за идентификацију пакета који чине једну целину.

Укупан број пакета је потребан у случају да се за пренос неке информације користи више од једног пакета, и показује од колико пакета се та информација састоји.

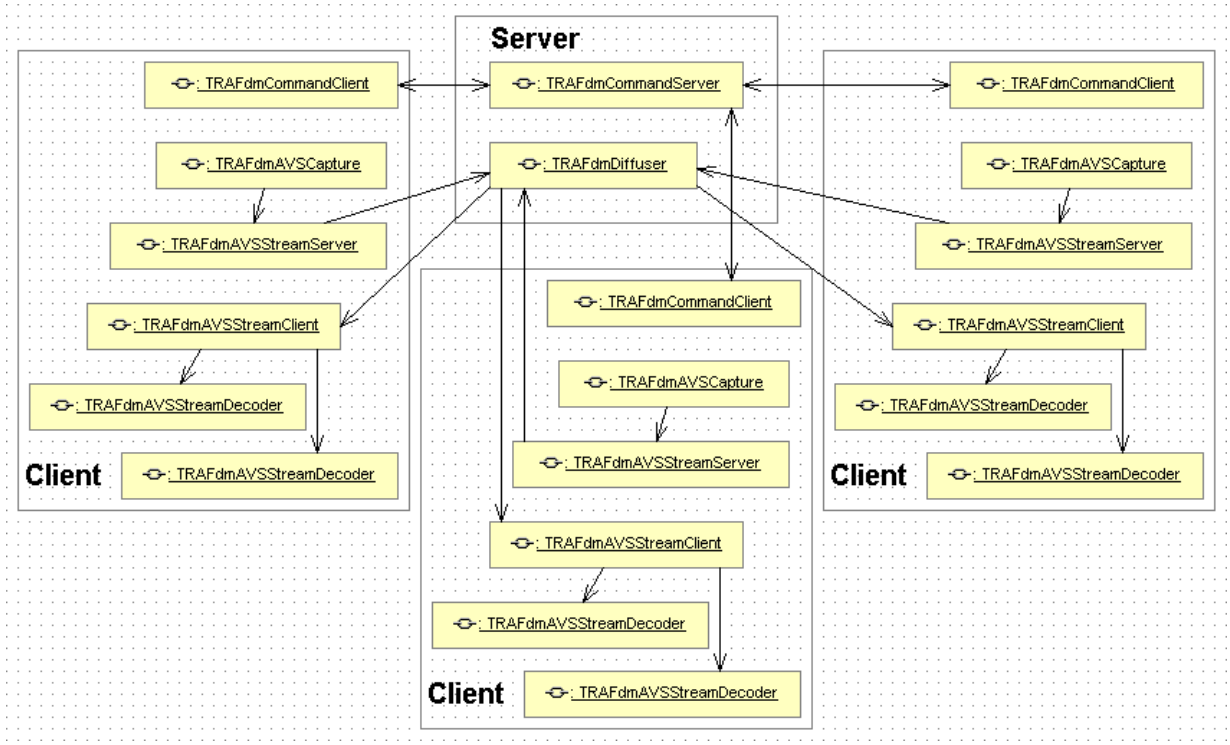
Индекс пакета је потребан у случају да се за пренос једног податка користи више од једног пакета, и показује редни број пакета у оквиру тог податка.

Величина поља података показује број бајтова који се преносе у пољу Подаци.

Поље Подаци је низ бајтова који садрже податке које треба пренети.

3.2.14 Пренос података кроз класе система

Следећи дијаграм приказује пренос података кроз класе система, у случају када предавање има три учесника. Један учесник је предавач, друга два су слушаоци. Одређености ради, узећемо да клијент приказан на левој страни дијаграма има улогу предавача.



Слика 24 - Ток података кроз класе система

Имамо два тока преноса података у систему. Једним током се преносе подаци командног протокола, а другим аудио и видео подаци и снимци екрана.

Пренос података између класа **`TRAFdmCommandClient`** и **`TRAFdmCommandServer`**, које ће имплементирати командни протокол, ће бити двосмерни, што нам је потребно за читање повратних вредности команди.

У преносу аудио и видео података и снимка екрана, разликујемо случајеве када је учесник у улози предавача и када је у улози слушаоца. Ако је учесник у улози предавача, уз његове аудио и видео податке се шаље и снимак екрана. Само предавач шаље снимак екрана и само слушаоци га примају.

Пренос аудио и видео података и снимка екрана почиње у класи **`TRAFdmAVSCapture`** прикупљањем података са микрофона и камере, а ако је учесник у улози предавача, снима се и тренутни садржај екрана. Аудио и видео подаци и снимак екрана предавача се преносе у класу **`TRAFdmAVSStreamServer`**, где се кодирају и компресују и шаљу преко рачунарске мреже класи **`TRAFdmDiffuser`**. Ова класа ће примљене податке проследити свим другим учесницима предавања. Прослеђене податке прима класа **`TRAFdmAVSStreamClient`**, и проследиће их одговарајућој инстанци класе **`TRAFdmAVSStreamDecoder`**, која те податке декомпресује и декодира, па се они могу репродуковати као звук или приказати као слика. У клијентском програму се за сваког учесника прави по једна инстанца класе **`TRAFdmAVSStreamDecoder`**.

3.2.15 Тестирање

Тестирање у првом циклусу ће бити усмерено само на испитивање да ли је исправност имплементације функционалности довољна за демонстрацију.

3.3 Писање програма

3.3.1 Конвенције имплементације

Без обзира који модел развоја користимо, веома је пожељно да се на почетку фазе имплементације обрати пажња на неке организационе проблеме важне за активност писања програма. Увешћемо неколико правила која ће нам олакшати разумевање, управљање и одржавање базе изворног кода.

Структура радног директоријума је следећа:

```
eLearning
|-runimage
|-runimage2
|-runimage3
|-runimage...
|-runimageN
|-source
 |-!dcu
```

eLearning је главни директоријум који садржи све друге директоријуме.

runimage је директоријум који ће садржати извршну структуру директоријума и све фајлове неопходне за рад програма. Фајлови из овог директоријума ће бити укључени у инсталациони програм и копирани на корисников рачунар приликом инсталације програма.

runimage2, **runimage3** и други слични, ако постоје, су тестни директоријуми, који ће садржати различите конфигурације програма. За потребе тестирања, ове директоријуме ћемо постављати као радне за програм, чиме ћемо мењати активну конфигурацију.

source је директоријум који ће садржати фајлове са изворним кодом.

!dcu је директоријум који ће садржати компајлиране фајлове који се повезују у извршни фајл.

Имена фајлова у којима је смештен изворни код ће бити облика **UNazivKlase**, тј. имаће префикс **U**. Ако је у једном фајлу дефинисано више класа, фајл се назива по најважнијој од њих.

Имена типова дефинисаних у програму ће бити облика **TRAFNazivTipa**, тј. имаће префикс **TRAF**. Имена класа дефинисаних у програму ће у свом имену укључивати имена класа које наслеђују, ако су те класе дефинисане у програму.

При писању изворног кода ћемо примењивати препоруке из документа *Стил Објектног Паскала* (Object Pascal Style Guide), који описује начин форматирања изворног кода и конвенције именовања других важних елемената изворног кода.

3.3.2 Дељени делови изворног кода

Имплементација серверског и клијентског дела система имају заједничке делове изворног кода. То су следеће датотеке:

URAF.pas
URAFfrm.pas

URAF.pas садржи иницијализациону рутину, заједничке типове, глобалне константе, променљиве и функције које се користе у свим деловима програма. Најважнији елементи програма дефинисани у овом фајлу су мрежни пакет (**TRAFUDPPacket**) и константе којима се дефинишу величина бафера за пренос података, мрежне адресе и портови и повратне вредности командног протокола.

URAFfrm.pas садржи класу TRAFfrm, која је базна класа за све прозоре у систему, и имплементира методе за позиционирање прозора и локализацију.

3.3.3 Сервер

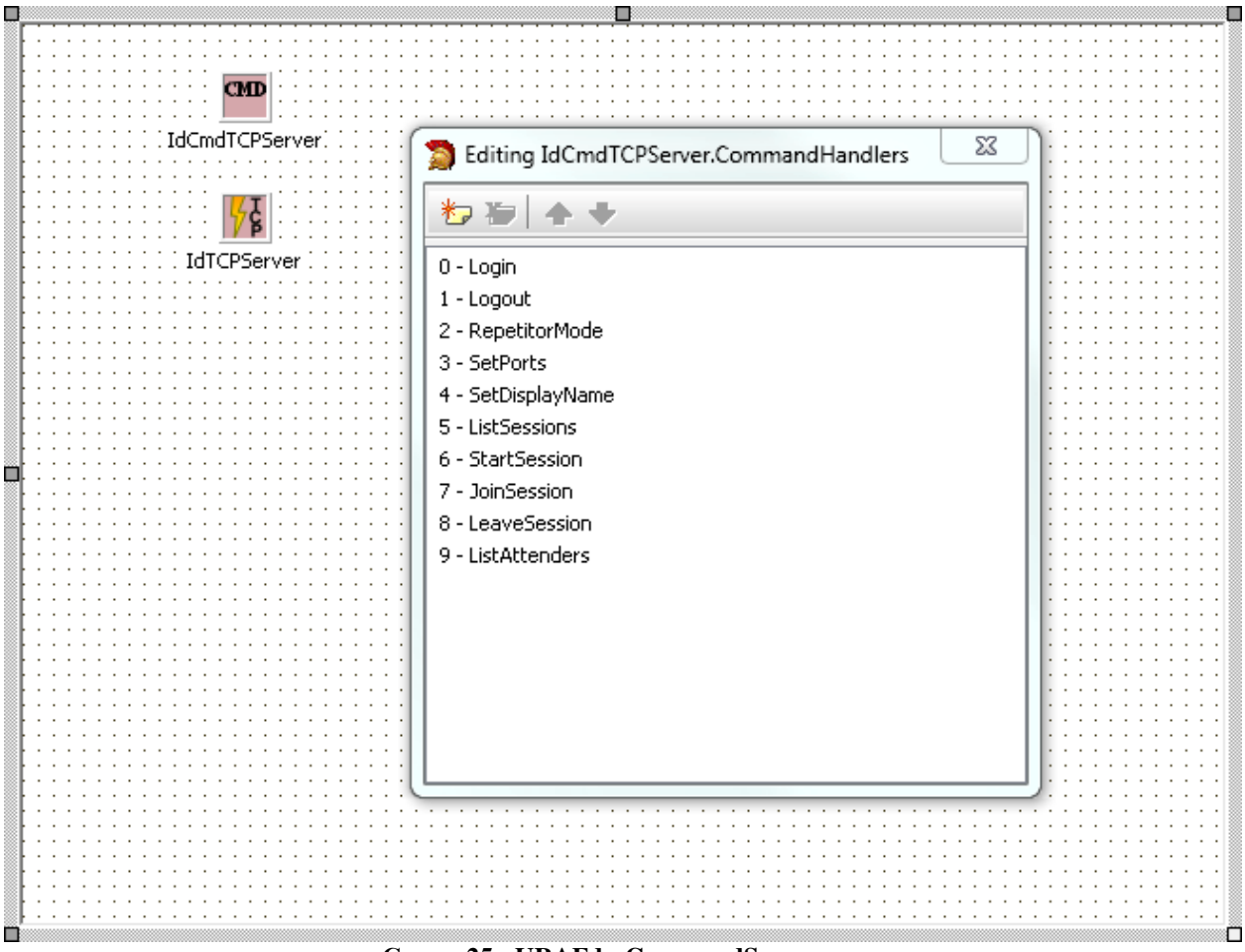
Сервер је имплементиран у следећим датотекама:

URAF.pas
URAFClients.pas
URAFdmCommandServer.pas
URAFdmDiffuser.pas
URAFfrm.pas
URAFfrmServer.pas
URAFSessions.pas

URAFClients.pas садржи класе **TRAFClient** и **TRAFClients**, које се користе за проверу идентитета клијената. Класа **TRAFClients** садржи листу корисника који могу да користе систем. У каснијим циклусима развоја, ова класа може бити надограђена да проверава идентитет клијената помоћу базе података.

URAFSessions.pas садржи класе **TRAFSessions**, **TRAFSession** и **TRAFSessionAttender** које се користе за управљање групама учесника.

URAFdmCommandServer.pas садржи класу **TRAFdmCommandServer** која имплементира серверску страну командног протокола. На слици је графички приказ овог модула у развојном окружењу и листа имплементираних команди.



Слика 25 - URAFdmCommandServer.pas

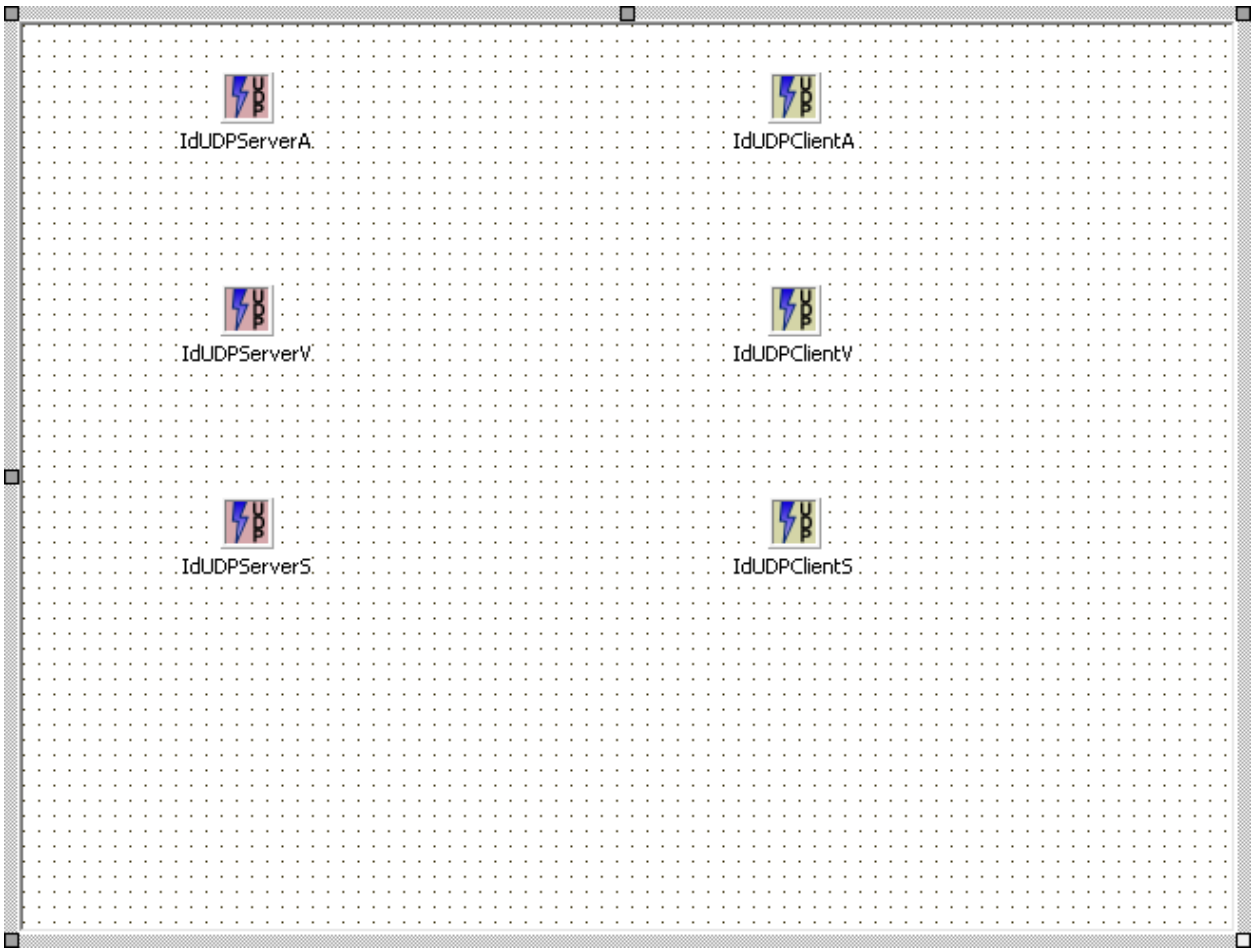
`IdCmdTCPServer` и **`IdTCPServer`** су компоненте из пакета *Инди Сокетс*. **`IdCmdTCPServer`** је типа **`TIdCmdTCPServer`** и имплементира вишенитни ТЦП сервер прилагођен за имплементацију текстуалних командних протокола. Прва веза коју клијент успоставља са сервером је управо веза са овом компонентом. **`IdTCPServer`** је типа **`TIdTCPServer`** и имплементира вишенитни ТЦП сервер опште намене.

По успостављању везе, креира се објекат типа **`TRAFSessionAttender`**, који ће постојати све време трајања везе. Овом објекту се додељује јединствени идентификациони број, повезује се са подацима о мрежној вези и клијенту се шаље поздравна порука (метода **`IdCmdTCPServerConnect`**).

На слици видимо и помоћни прозор за дефинисање команди и листу дефинисаних команди. За сваку од команди је написана метода за обраду (методе чија су имена облика **`IdCmdTCPServerCommandHandlers*Command`**, где је * број). Током извршавања метода за обраду команди, проверава се исправност контекста, и ако је контекст исправан, команда се извршава. Повратна вредност команде означава да ли је команда успешно извршена или није. Ако команда није успешно извршена, резултат означава разлог. Све повратне вредности команди су дефинисане у датотеци **`URAF.pas`**.

Компонента **`IdTCPServer`** се користи за обавештавање клијената о догађајима на серверу попут придруживања или напуштања групе и размену текстуалних порука.

`URAFdmDiffuser.pas` дефинише класу **`TRAFdmDiffuser`** која се користи за пријем и слање аудио и видео података између учесника. На слици је графички приказ овог модула у развојном окружењу.

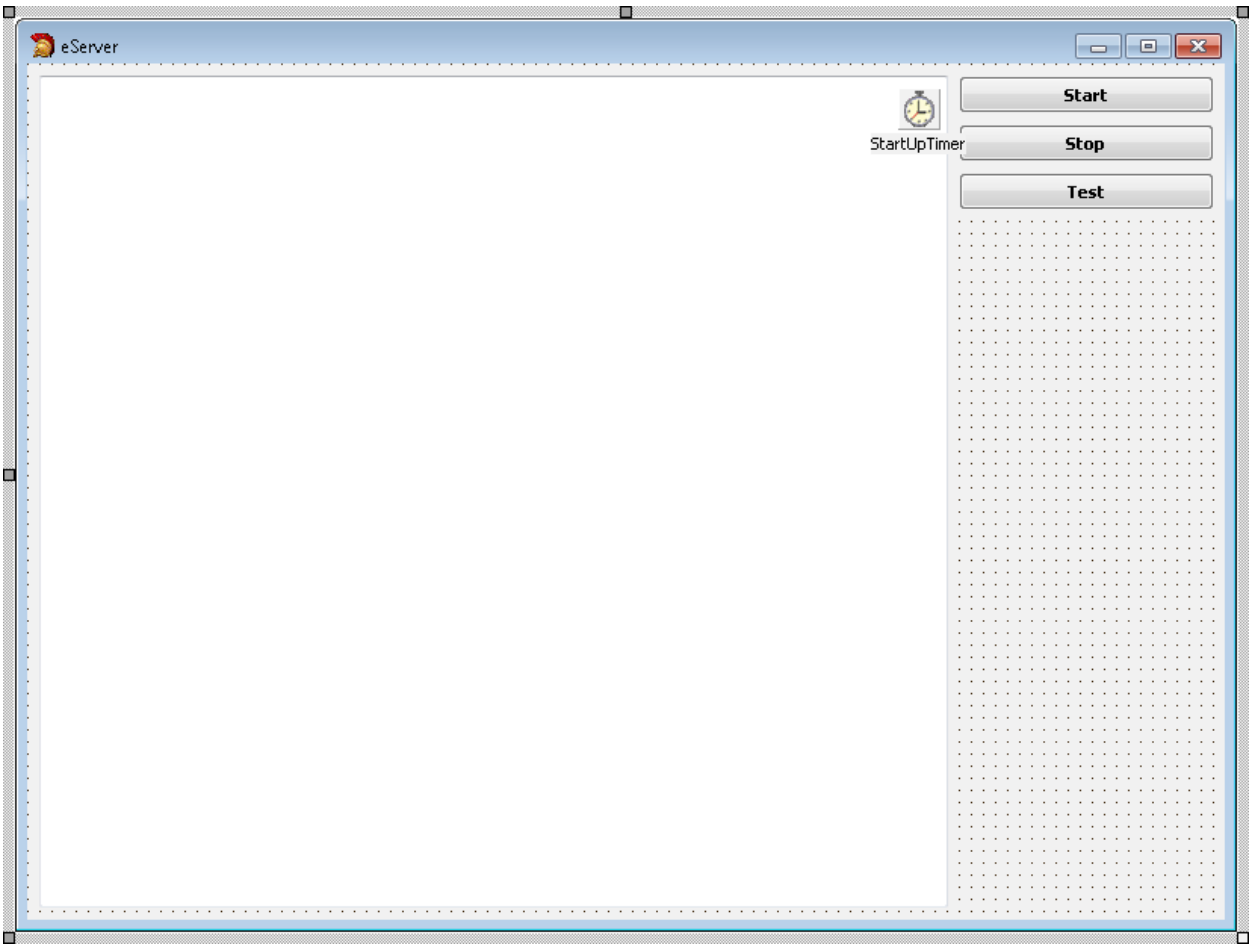


Слика 26 - URAFdmDiffuser.pas

Све приказане компоненте припадају пакету *Инди Сокетс*. Компоненте **`IdUDPServerA`**, **`IdUDPServerV`** и **`IdUDPServerS`** су типа **`TIdUDPServer`** и имплементирају УДП сервер опште намене. Компоненте **`IdUDPClientA`**, **`IdUDPClientV`** и **`IdUDPClientS`** су типа **`TIdUDPClient`** и имплементирају УДП клијента опште намене.

Постоје три пара компоненти сервера и клијента, и сваки пар је задужен за пренос једне врсте сигнала – А за аудио, В за видео и С за снимак екрана. За сваку компоненту типа **`TIdUDPServer`** је имплементирана метода за догађај **`OnUDPRead`**, која се извршава одмах након пријема података од клијента (методе **`IdUDPServerAUDPRead`**, **`IdUDPServerVUDPRead`**, **`IdUDPServerSUDPRead`**). Сви клијенти шаљу своје податке компонентама **`IdUDPServerA`**, **`IdUDPServerV`** и **`IdUDPServerS`**. Када нека серверска компонента прими мрежни пакет од клијента, у методи за **`OnUDPRead`** догађај се анализира заглавље пакета, утврђује се учесник пошиљалац и његова група, и онда се тај мрежни пакет, преко клијентских компоненти, прослеђује свим члановима те групе корисника, без пошиљаоца.

`URAFfrmServer.pas` дефинише класу **`TRAFfrmServer`** која имплементира главни прозор корисничког интерфејса сервера. На слици је графички приказ овог модула у развојном окружењу.



Слика 27 - URAFfrmServer.pas

Као што смо раније навели, сервер треба да буде израђен као сервисна апликација, али ћемо га у почетним циклусима развоја, ради лакшег тестирања, израдити као корисничку апликацију. Због тога, графички кориснички интерфејс нема велики значај. Интерфејс је веома једноставан и укључује неколико контролних дугмића и простор за испис дијагностичких порука. Потпуно је раздвојен од кључних делова система што ће олакшати прераду из корисничке у сервисну апликацију.

3.3.4 Клијент

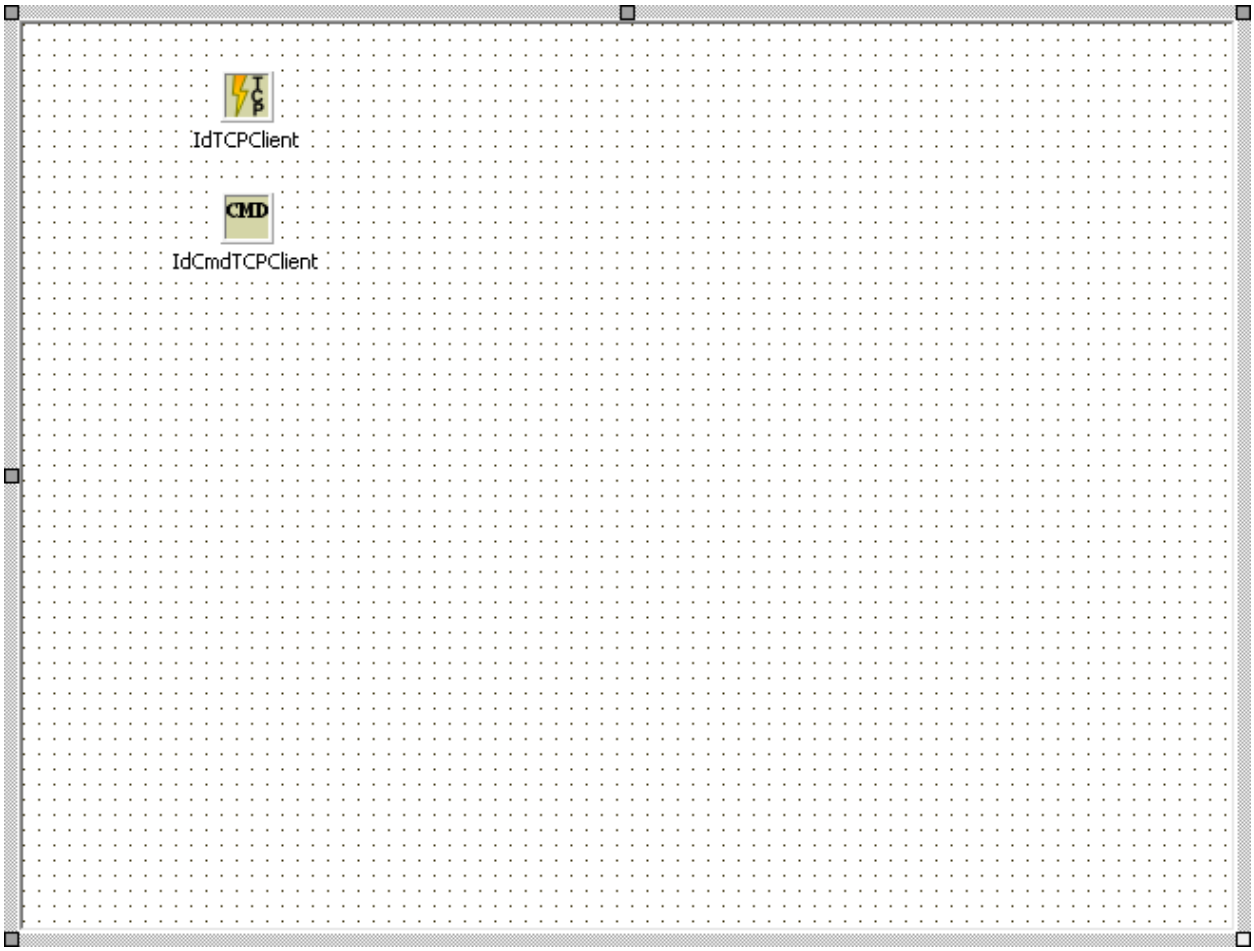
Клијент је имплементиран у следећим датотекама:

URAF.pas
URAFdmAVSCapture.pas
URAFdmAVSStreamClient.pas
URAFdmAVSStreamDecoder.pas
URAFdmAVSStreamServer.pas
URAFdmCommandClient.pas
URAFdmResources.pas
URAFfrm.pas
URAFfrmAttenderFrame.pas
URAFfrmClient.pas
URAFfrmDebug.pas
URAFfrmEd.pas
URAFfrmEdHandwritingRecognizer.pas
URAFfrmEdPDF.pas
URAFfrmEdPresentersScreen.pas

URAFfrmEdTextEditor.pas
URAFfrmEdTPCWhiteboard.pas
URAFfrmEdWebBrowser.pas
URAFfrmEdWhiteboard.pas
URAFfrmJoinSessions.pas
URAFfrmLogin.pas
URAFPenMoves.pas
URAFPerfCounter.pas

Појединачне модуле нећемо описивати редоследом који су наведени, већ по логичком редоследу извршавања операција у програму. Прво ћемо описивати компоненте које генеришу и обрађују податке, а затим кориснички интерфејс.

URAFdmCommandClient.pas дефинише класу **TRAFdmCommandClient** која имплементира клијентску страну командног протокола. На слици је графички приказ овог модула у развојном окружењу.



Слика 28 - URAFdmCommandClient.pas

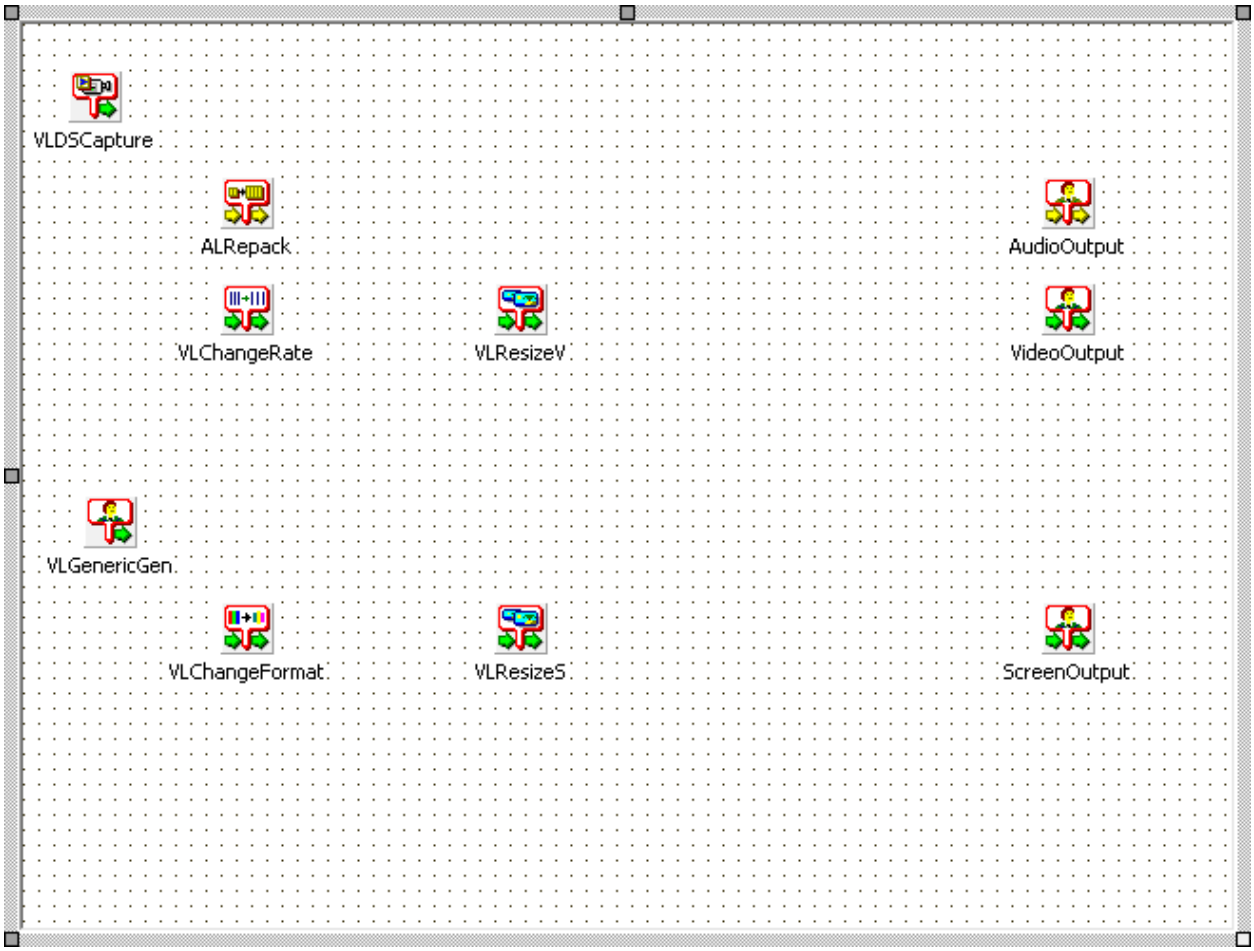
IdTCPClient и **IdCmdTCPClient** су компоненте из пакета *Инди Сокетс*. **IdTCPClient** је типа **TIdTCPClient** и имплементира ТЦП клијента опште намене. Прва веза коју клијент успоставља са сервером се успоставља преко ове компоненте. **IdCmdTCPClient** је типа **TIdCmdTCPClient** и имплементира ТЦП клијента са могућношћу пријема команди од сервера.

Ове две компоненте успостављају везу са компонентама **IdCmdTCPServer** и **IdTCPServer** из класе **TRAFdmCommandServer** у серверу, и то **IdTCPClient** са

IdCmdTCPServer и **IdCmdTCPClient** са **IdTCPServer**. Компонента **IdTCPClient** се користи за издавање команди серверу, а компонента **IdCmdTCPClient** за пријем обавештења о догађајима на серверу попут придруживања или напуштања групе.

Команде се издају позивањем метода класе, а за сваку команду командног протокола постоји одговарајућа метода истог имена. Методе као резултат враћају структуру података која садржи одговор сервера (**TIdReply**). Класа се не бави провером исправности контекста издавања команди по коначном аутомату, нити анализом резултата извршавања. За то је задужен део програма који позива методу.

URAFdmAVSCapture.pas дефинише класу **TRAFdmAVSCapture** која имплементира прикупљање аудио и видео података и снимка екрана. На слици је графички приказ овог модула у развојном окружењу.



Слика 29 - URAFdmAVSCapture.pas

Све приказане компоненте припадају пакетима *Митов Аудио Лаб* и *Митов Видео Лаб*. **VLDSCapture** је типа **TVLDSCapture** и имплементира прикупљање аудио и видео података са камере и микрофона помоћу *Дајрект Шоу* (DirectShow) технологије. **VLGenericGen** је типа **TVLGenericGen** и имплементира периодично генерисање фрејмова. **ALRepack** је типа **TALRepack** и имплементира промену формата аудио сигнала (нивои и фреквенција узорковања и број и врста канала). **VLChangeRate** је типа **TVLChangeRate** и имплементира промену броја фрејмова видео сигнала. **VLChangeFormat** је типа **TVLChangeFormat** и имплементира промену формата слике. **VLResizeV** и **VLResizeS** су типа **TVLResize** и имплементирају промену димензија фрејмова. **AudioOutput** је типа **TALGenericFilter** и имплементира генерички

филтер за обраду аудио података. **VideoOutput** и **ScreenOutput** су типа **TVLGenericFilter** и имплементирају генерички филтер за обраду видео података.

Компоненте су, преко својих својстава **InputPin** и **OutputPin**, повезане у три комуникациона канала, по један за сваку врсту података. Канал за аудио податке чине компоненте **VLDSCapture**, **ALRepack** и **AudioOutput**. Канал за видео податке чине компоненте **VLDSCapture**, **VLChangeRate**, **VLResizeV** и **VideoOutput**. Канал за снимак екрана чине компоненте **VLGenericGen**, **VLChangeFormat**, **VLResizeS** и **ScreenOutput**.

Аудио подаци се прикупљају са микрофона помоћу компоненте **VLDSCapture**. У компоненти **ALRepack** се прилагођава формат аудио података. Компонента **AudioOutput** је излазни чвор аудио података који су прошли обраду. Сви други делови програма се повезују на ову компоненту и преко ње добијају аудио податке. Прилагођавањем формата аудио сигнала се утиче на квалитет звука који се преноси.

Видео подаци се прикупљају са камере помоћу компоненте **VLDSCapture**. У компоненти **VLChangeRate** се прилагођава број фрејмова, а у компоненти **VLResizeV** се прилагођавају димензије фрејма. Компонента **VideoOutput** је излазни чвор видео података. Сви други делови програма се повезују на ову компоненту и преко ње добијају видео податке. Прилагођавањем броја и димензија фрејмова се утиче на квалитет видео снимка који се преноси.

Снимци екрана се прикупљају помоћу компоненте **VLGenericGen**, тако што се периодично приступа тзв. графичком контексту прозора радне површине оперативног система, део његовог садржаја се копира у битмап слику и претвара у облик погодан за пренос кроз комуникациони канал компоненти (метода **VLGenericGenGenerate**). На излазу компоненте **VLGenericGen** добијамо фрејм који садржи потребан део снимка екрана. У компоненти **VLChangeFormat** се прилагођава формат, тј. број боја фрејма. У компоненти **VLResizeS** се прилагођавају димензије фрејма. Компонента **ScreenOutput** је излазни чвор снимка екрана. Сви други делови програма који раде са снимком екрана се повезују на ову компоненту и преко ње добијају податке. Прилагођавањем формата и димензија фрејмова се утиче на квалитет снимка екрана који се преноси. Додатно, прилагођавањем формата се и елиминишу неки проблеми који настају услед различитих видео конфигурација клијентских рачунара.

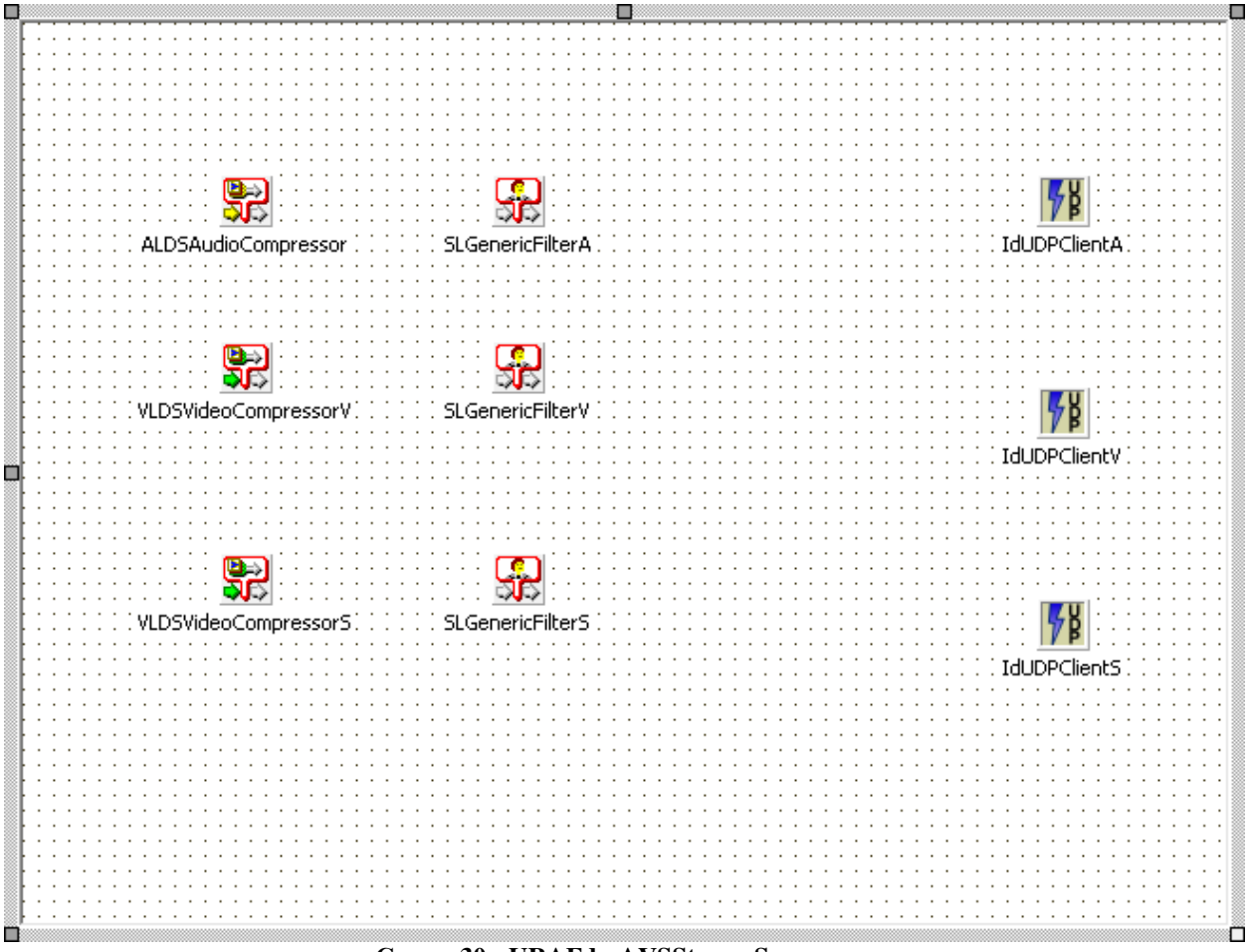
Снимци екрана се прикупљају само ако је учесник у улози предавача.

Аудио сигнал се узоркује фреквенцијом од 11025 Hz, двоканално и на 16 битне вредности (65536 нивоа). Видео сигнал се узоркује на 30 фрејмова по секунди, у 24 битним бојама (RGB24), димензија 320x240 пиксела. Конвертује се на 5 фрејмова по секунди, и димензије се смањују на 120x90 пиксела. Снимак екрана се узоркује на 5 фрејмова по секунди, у 32 битним бојама (RGBA32), димензија 800x600.

Интензивном употребом софтверских компоненти, сложени процес прикупљања и обраде аудио и видео података и снимка екрана је веома једноставно реализован. Модул садржи веома мало писаног кода, а једини процес који је тако реализован је преузимање снимка екрана.

URAFdmAVSStreamServer.pas дефинише класу **TRAFdmAVSStreamServer** која имплементира кодирање и компресовање аудио и видео података и снимка екрана и слање

ових података посредничком серверу. На слици је графички приказ овог модула у развојном окружењу.



Слика 30 - URAFdmAVSStreamServer.pas

Све приказане компоненте припадају пакетима *Митов Аудио Лаб* , *Митов Видео Лаб, Митов Сигнал Лаб* и *Инди Сокетс*. **ALDSAudioCompressor** је типа **TALDSAudioCompressor** и имплементира кодирање и компресовање аудио података помоћу *Дајрект Шоу* технологије. **VLDSVideoCompressorV** и **VLDSVideoCompressorS** су типа **TVLDSVideoCompressor** и имплементирају кодирање и компресовање видео података помоћу *Дајрект Шоу* технологије. **SLGenericFilterA**, **SLGenericFilterV** и **SLGenericFilterS** су типа **TSLGenericFilter** и имплементирају генерички филтер за обраду података. **IdUDPClientA**, **IdUDPClientV** и **IdUDPClientS** су типа **TIdUDPClient** и имплементирају УДП клијента опште намене.

Слично као код **TRAFdmAVSCapture**, компоненте су, преко својих својстава **InputPin** и **OutputPin**, повезане у три комуникациона канала, по један за сваку врсту података. Канал за аудио податке чине компоненте **ALDSAudioCompressor**, **SLGenericFilterA** и **IdUDPClientA**. Канал за видео податке чине компоненте **VLDSVideoCompressorV**, **SLGenericFilterV** и **IdUDPClientV**. Канал за снимак екрана чине компоненте **VLDSVideoCompressorS**, **SLGenericFilterS** и **IdUDPClientS**.

Додатно, компоненте **ALDSAudioCompressor**, **VLDSVideoCompressorV** и **VLDSVideoCompressorS** су, преко својих својстава **InputPin** повезане на компоненте

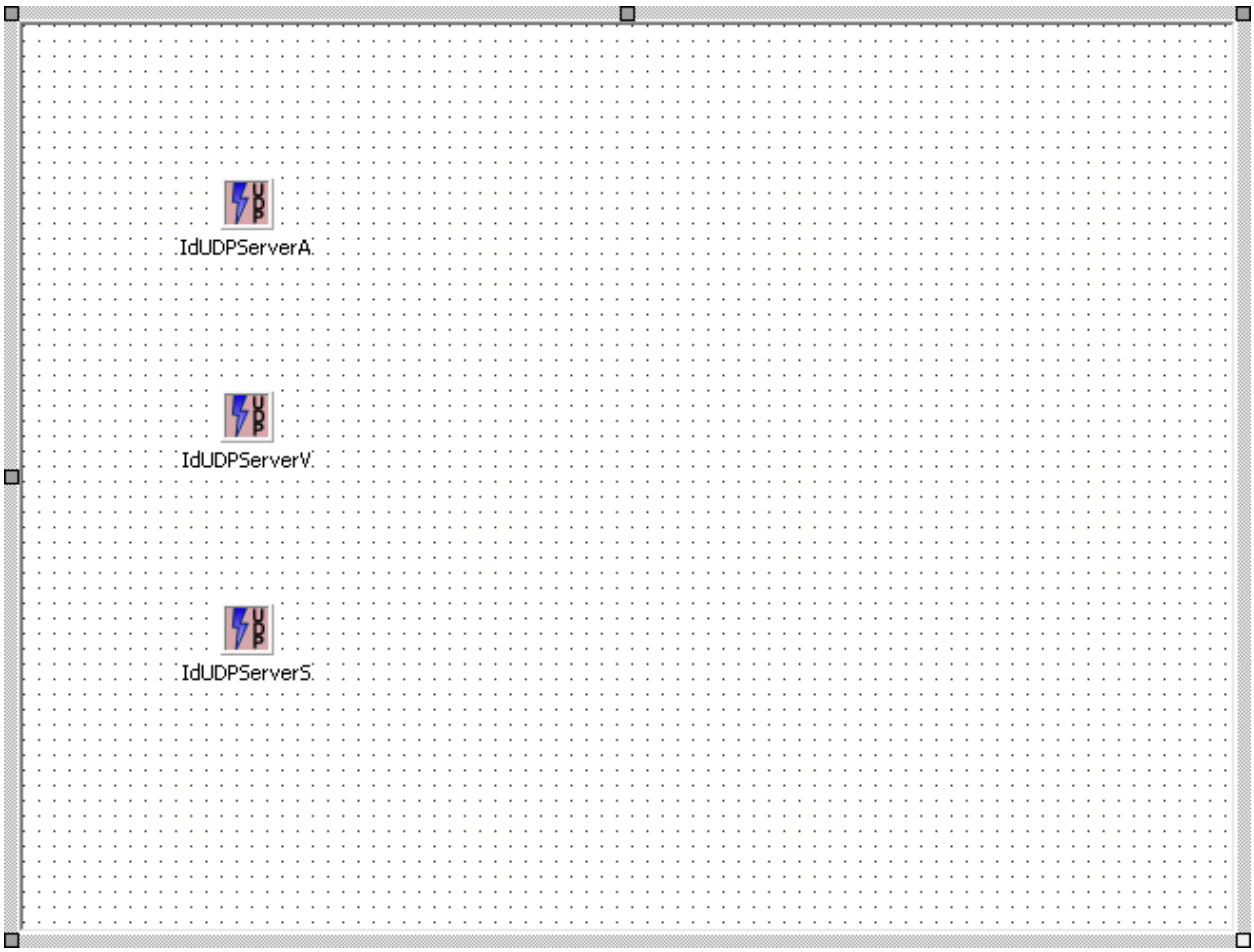
AudioOutput, **VideoOutput** и **ScreenOutput** из класе **TRAFdmAVSCapture**, од које преузимају податке које треба да обраде.

Начин обраде података је сличан за сваки комуникациони канал. Подаци се преузму са одговарајућег излазног чвора класе **TRAFdmAVSCapture**, кодирају и компресују се изабраним аудио или видео кодеком и преко **IdUDPClient** компоненти се шаљу посредничком серверу. Подаци се шаљу серверу УДП протоколом, у датаграмима највеће величине нешто мање од 64 килобајта, по јединицама обраде (звучном узорку или фрејму). Уколико је количина података у јединици обраде већа од 64 килобајта подаци се деле у више пакета.

Аудио подаци се претварају у једноканалне и кодирају и компресују CCITT μ-Law алгоритмом. Видео подаци и снимак екрана се кодирају и компресују MPEG-4 алгоритмом.

Компоненте типа **TIdUDPClient** шаљу податке одговарајућим компонентама типа **TIdUDPServer**, из класе **TRAFdmDiffuser** из сервера, и то **IdUDPClientA** ка **IdUDPServerA**, **IdUDPClientV** ка **IdUDPServerV** и **IdUDPClientS** ка **IdUDPServerS**.

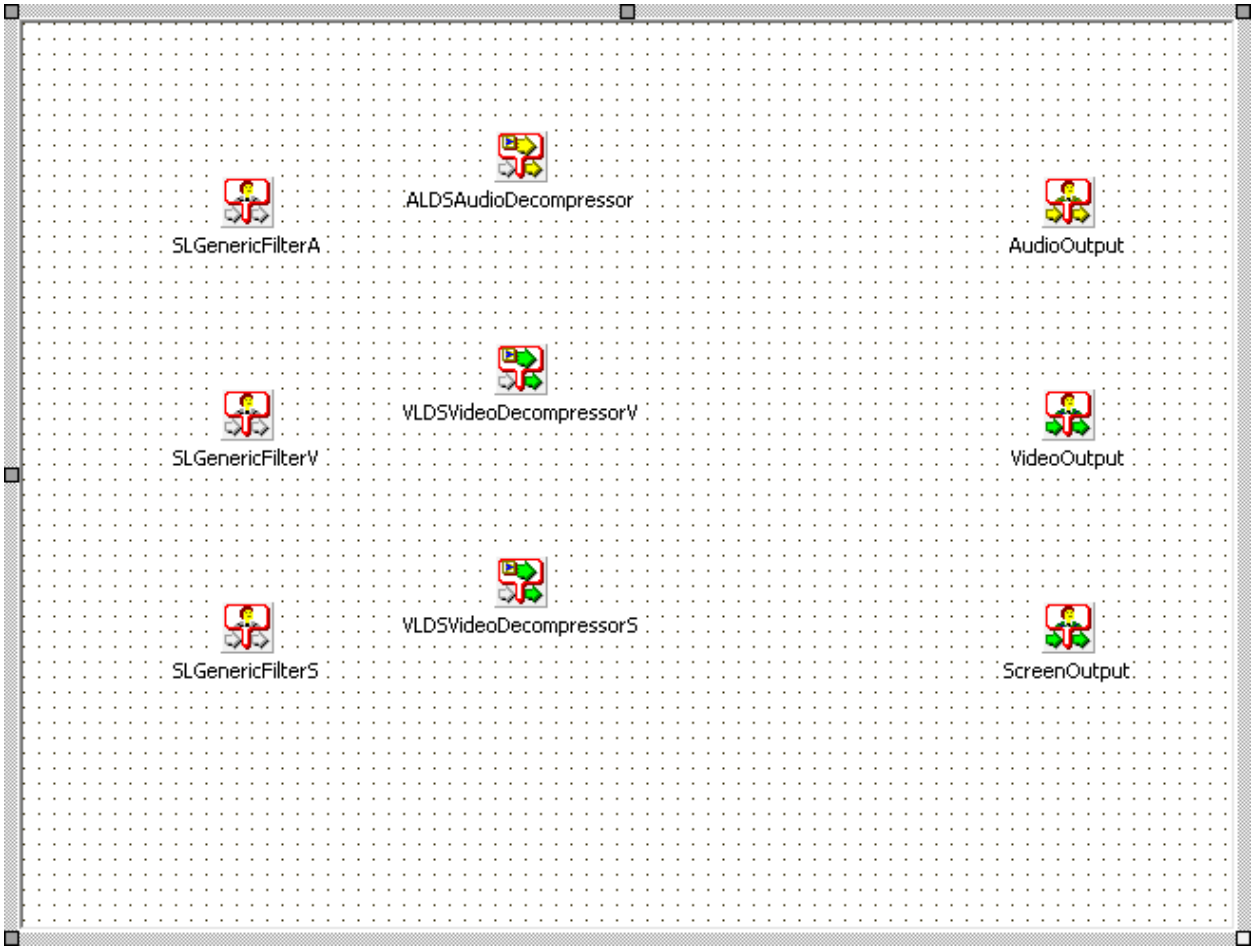
URAFdmAVSStreamClient.pas дефинише класу **TRAFdmAVSStreamClient** која имплементира пријем аудио и видео података и снимка екрана од посредничког сервера. На слици је графички приказ овог модула у развојном окружењу.



Слика 31 - URAFdmAVSStreamClient.pas

Све приказане компоненте припадају пакету *Инди Сокетс*. **IdUDPServerA**, **IdUDPServerV** и **IdUDPServerS** су типа **TIdUDPServer** и имплементирају УДП сервер опште намене.

URAFdmAVSStreamDecoder.pas дефинише класу **TRAFdmAVSStreamDecoder** која имплементира декомпресовање и декодирање аудио и видео података и снимка екрана. На слици је графички приказ овог модула у развојном окружењу.



Слика 32 - URAFdmAVSStreamDecoder.pas

Све приказане компоненте припадају пакетима *Митов Аудио Лаб, Митов Видео Лаб* и *Митов Сигнал Лаб*. **SLGenericFilterA**, **SLGenericFilterV** и **SLGenericFilterS** су типа **TSLGenericFilter** и имплементирају генерички филтер за обраду података. **ALDSAudioDecompressor** је типа **TALDSAudioDecompressor** и имплементира декомпресовање и декодирање аудио података помоћу *Дајрект Шоу* технологије. **VLDSVideoDeompressorV** и **VLDSVideoDecompressorS** су типа **TVLDSVideoDecompressor** и имплементирају декомпресовање и декодирање видео података помоћу *Дајрект Шоу* технологије. **AudioOutput** је типа **TALGenericFilter** и имплементира генерички филтер за обраду аудио података. **VideoOutput** и **ScreenOutput** су типа **TVLGenericFilter** и имплементирају генерички филтер за обраду видео података.

Класе **TRAFdmAVSStreamClient** и **TRAFdmAVSStreamDecoder** су међусобно зависне и чине функционалну целину и зато их разматрамо заједно.

Класа **TRAFdmAVSStreamClient** за сваког учесника предавања прави инстанцу класе **TRAFdmAVSStreamDecoder**, и чува их у листи (члан **FDecoders**). Једна инстанца декодера обрађује податке за само једног учесника предавања.

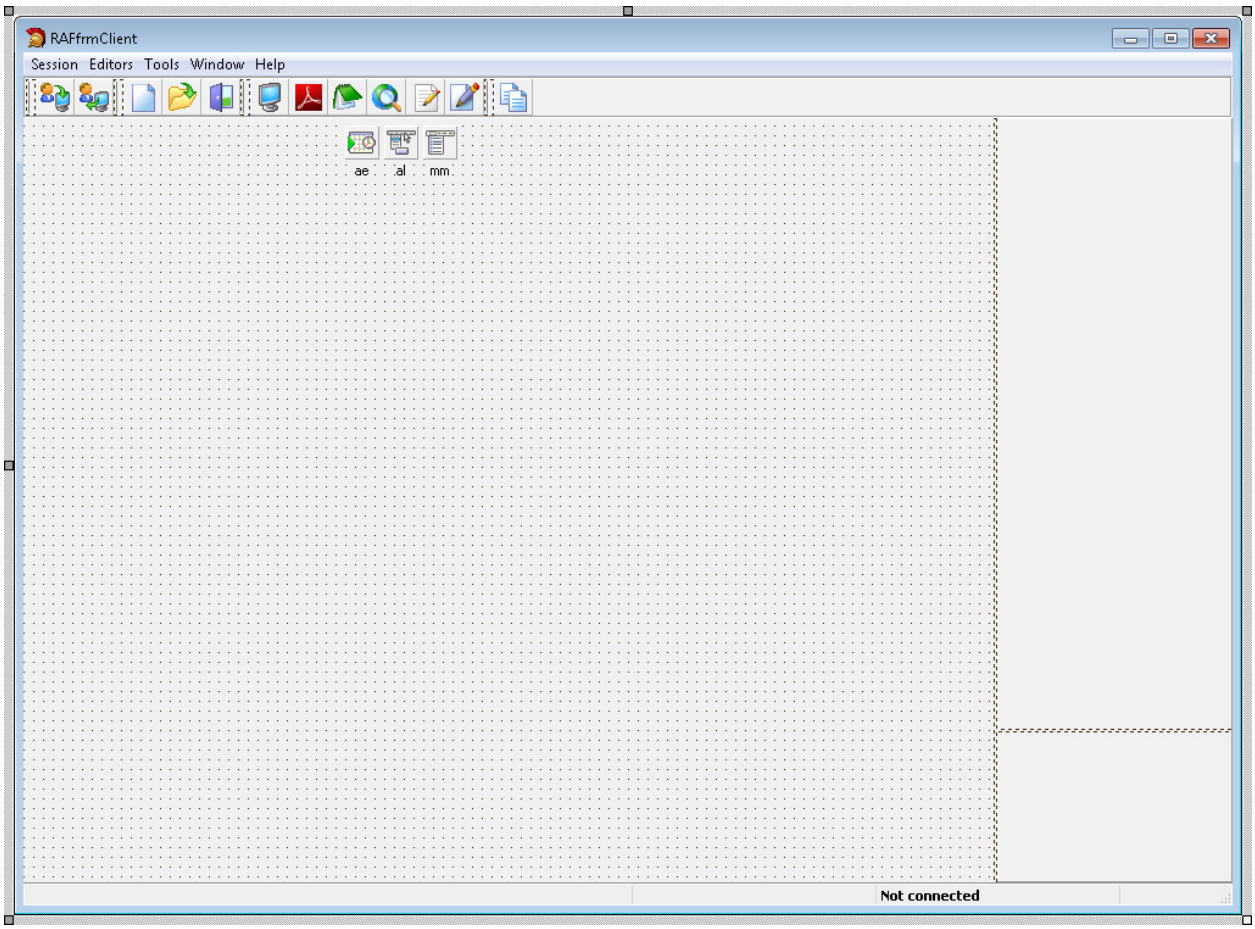
И овде су компоненте груписане у три комуникациона канала, по један за сваки тип података. Канал за аудио податке чине компоненте **IdUDPServerA**, **SLGenericFilterA**, **ALDSAudioDecompressor** и **AudioOutput**. Канал за видео податке чине компоненте **IdUDPServerV**, **SLGenericFilterV**, **VLDSVideoDecompressorV** и **VideoOutput**. Канал за снимак екрана чине компоненте **IdUDPServerS**, **SLGenericFilterS**, **VLDSVideoCompressorS** и **IdUDPClientS**.

Компоненте типа **TIdUDPServer** из класе **TRAFdmAVSStreamClient** примају податке од одговарајућих компоненти типа **TIdUDPClient** из класе **TRAFdmDiffuser** из сервера, и то **IdUDPServerA** од **IdUDPClientA**, **IdUDPServerV** од **IdUDPClientV** и **IdUDPServerS** од **IdUDPClientS**. Слично као и у серверу, за сваку компоненту типа **TIdUDPServer** је имплементирана метода за догађај **OnUDPRead**, која се извршава одмах након пријема података од клијента (методе **IdUDPServerAUDPRead**, **IdUDPServerVUDPRead**, **IdUDPServerSUDPRead**). Када нека од компоненти прими мрежни пакет од сервера, у методи за **OnUDPRead** догађај се анализира заглавље пакета, утврђује се пошиљалац и тај мрежни пакет се прослеђује декодеру који обрађује податке за тог пошиљаоца.

Мрежни пакет се прослеђује декодеру преко метода **ProcessA**, **ProcessV** и **ProcessS**. Мрежни пакет се у овим методама претвара у облик погодан за пренос кроз комуникациони канал компоненти, и помоћу компоненти типа **TSLGenericFilter** се шаље декомпресору. На излазу декомпресора се добијају аудио и видео подаци и снимак екрана и даље се воде на излазне чворове **AudioOutput**, **VideoOutput** и **ScreenOutput** одакле ће их преузимати делови програма којима су ови подаци потребни.

3.3.5 Кориснички интерфејс клијента

URAFfrmClient.pas дефинише класу **TRAFfrmClient** која имплементира главни прозор графичког корисничлког интерфејса. На слици је графички приказ овог модула у развојном окружењу.



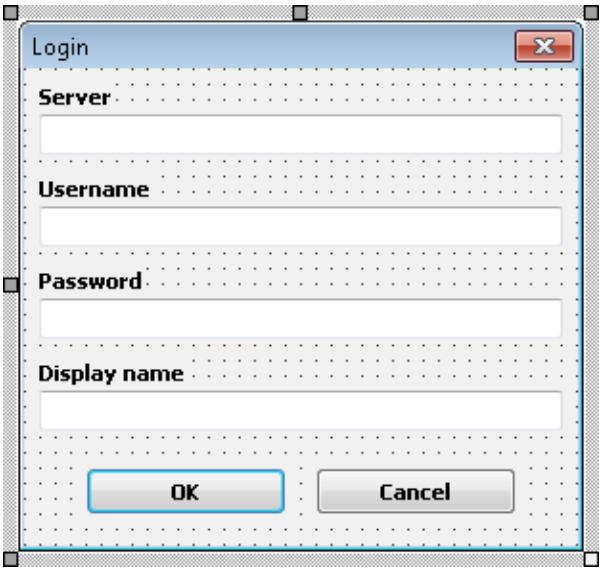
Слика 33 - URAFfrmClient.pas

Главни прозор корисничког интерфејса је имплементиран у складу са планом, основа је тзв. *МДИ интерфејс*, коме су додати мени и палета алатки преко којих се позивају команде програма, централни део је простор у коме ће бити приказивани подпрозори, а десно је простор резервисан за приказивање видео снимака учесника. Са простора у коме се приказују подпрозори се узима снимак екрана, када клијент ради у режиму предавача. Елементи корисничког интерфејса се правилима коначног аутомата прилагођавају тако да није могуће позвати команду чије издавање нема смисла у тренутном контексту. Додата је и могућност да се садржај тренутно активног подпорзора постави као позадина на таблу за цртање, чиме се омогућава да предавач означава важне делове или црта преко материјала који приказује.

Дугмићи на палети алатки, с лева на десно, имају следеће функције:

1. Пријава на систем
2. Одјављивање са система
3. Прављење нове групе и узимање улоге предавача
4. Придруживање групи и узимање улоге слушаоца
5. Напуштање групе
6. Позивање подпрозора типа **`TRAFfrmEdPresentersScreen`**
7. Позивање подпрозора типа **`TRAFfrmEdPDFReader`**
8. Позивање подпрозора типа **`TRAFfrmEdTextEditor`**
9. Позивање подпрозора типа **`TRAFfrmEdWebBrowser`**
10. Позивање подпрозора типа **`TRAFfrmEdHandwritingRecognizer`**
11. Позивање подпрозора типа **`TRAFfrmEdTPCWhiteboard`**
12. Копирање садржаја тренутно активног подпрозора на таблу за цртање

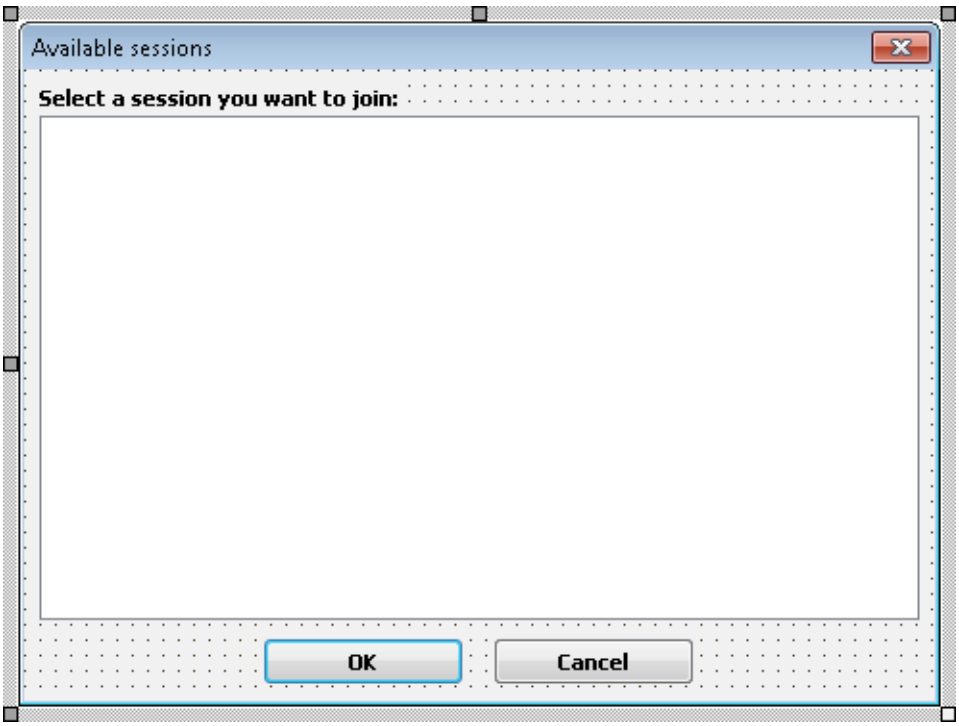
URAFfrmLogin.pas дефинише класу **TRAFfrmLogin** која имплементира помоћни дијалог за унос корисничког имена и лозинке. На слици је графички приказ овог модула у развојном окружењу.



Слика 34 - URAFfrmLogin.pas

Дијалог садржи поља за унос података потребних за пријављивање на систем.

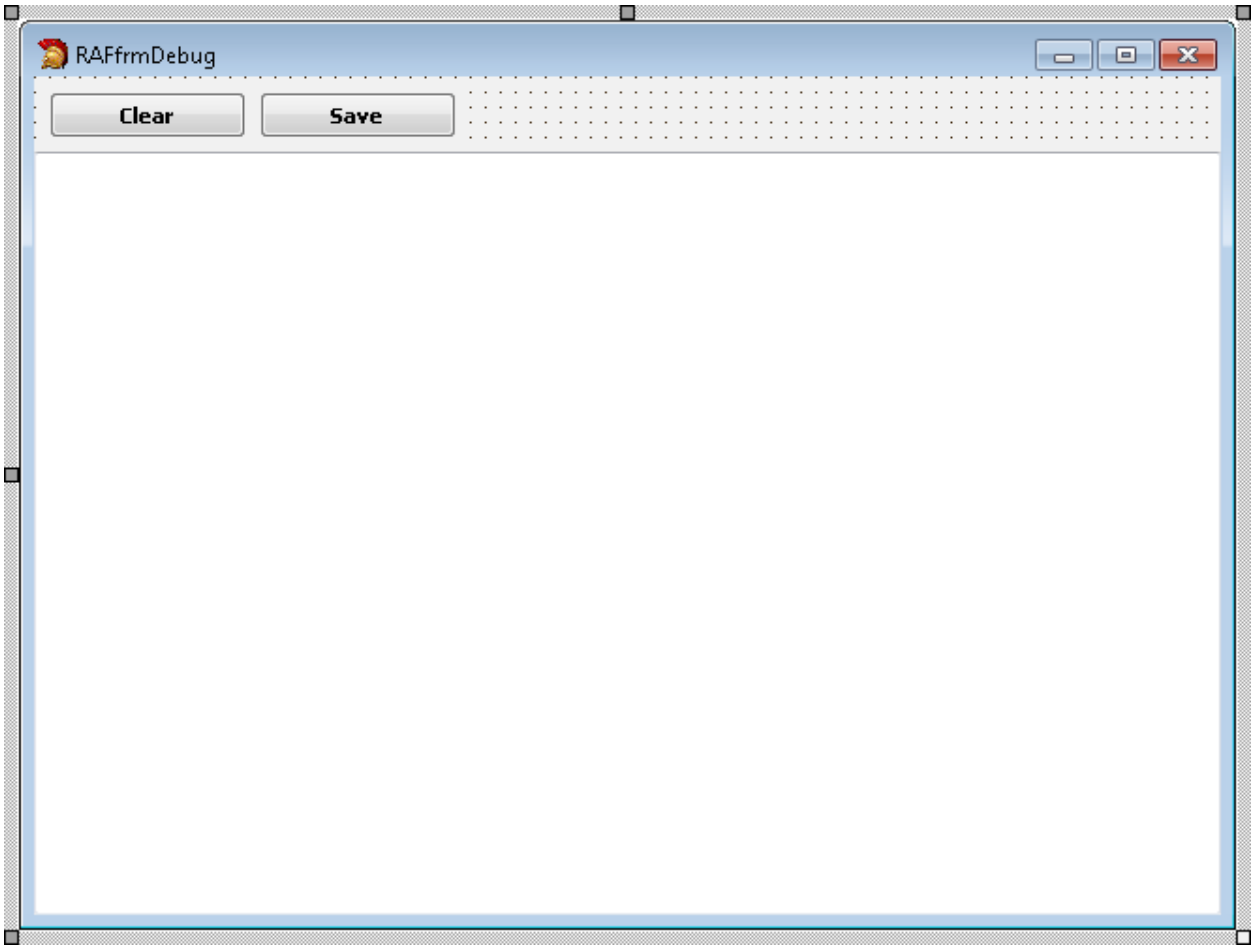
URAFfrmJoinSession.pas дефинише класу **TRAFfrmJoinSession** која имплементира помоћни дијалог преко кога корисник бира групу којој ће се придружити. На слици је графички приказ овог модула у развојном окружењу.



Слика 35 - URAFfrmJoinSession.pas

Дијалог садржи листу у којој ће бити приказана имена тренутно постојећих група.

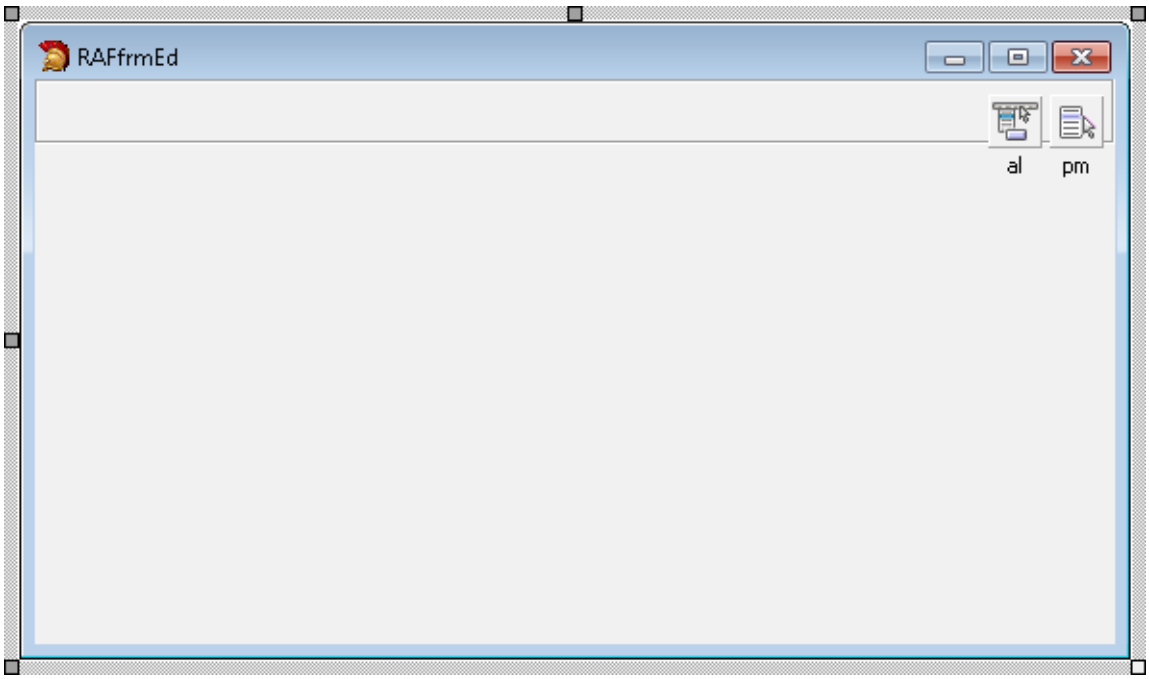
URAFfrmDebug.pas дефинише класу **TRAFfrmDebug** која имплементира помоћни дијалог за приказ дијагностишких порука. На слици је графички приказ овог модула у развојном окружењу.



Слика 36 - URAFfrmDebug.pas

Дијалог садржи простор за исписивање дијагностичких порука и дугмиће за брисање записника и снимање записника у текстуални фајл.

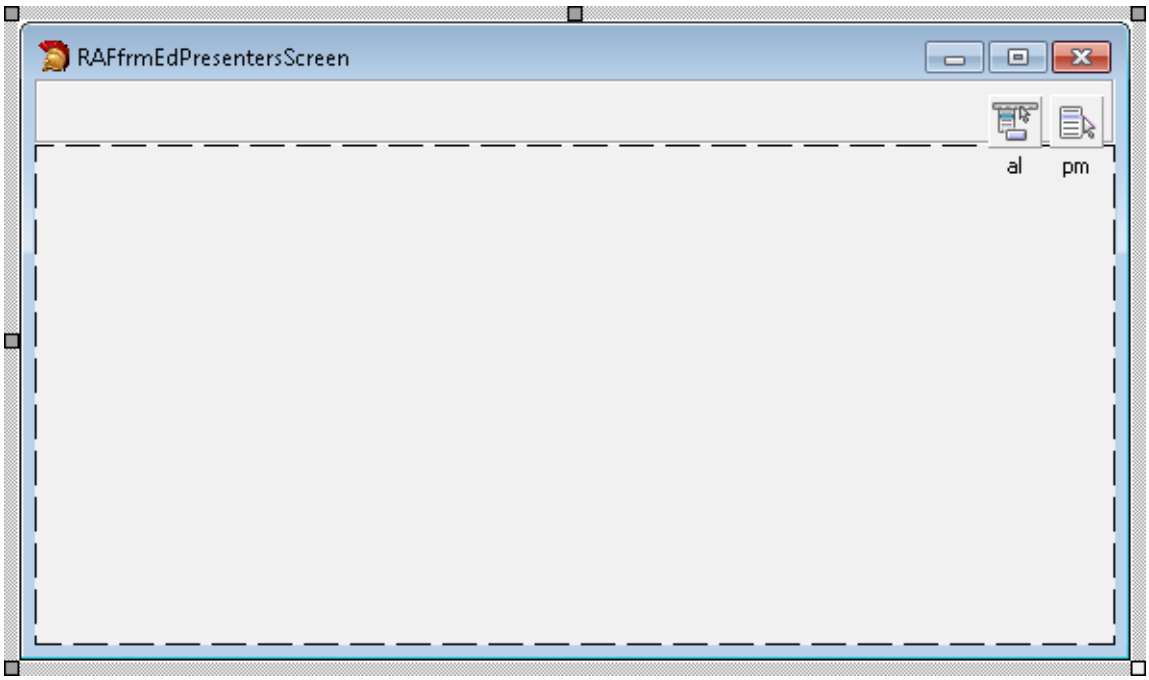
URAFfrmEd.pas дефинише класу **TRAFfrmEd** која је базна класа за све подпрозоре за приказ наставног материјала. На слици је графички приказ овог модула у развојном окружењу.



Слика 37 - URAFfrmEd.pas

Класа **`TRAFfrmEd`** садржи виртуелне методе које су значајне за представљање наставног материјала (**`LoadFromFile`** и **`LoadFromStream`**) и методу за снимање садржаја прозора у битмап слику (**`GetBitmap`**). Такође садржи и листу акција у којој ћемо дефинисати акције специфичне за приказивање конкретне врсте наставног материјала, и искачући мени и палету алатки које ћемо користити да прикажемо дефинисане акције. Ако палета алатки не садржи ни једно дугме, неће бити приказана.

`URAFfrmEdPresentersScreen.pas` дефинише класу **`TRAFfrmEdPresentersScreen`** која имплементира подпрозор за приказ снимка екрана предавача. На слици је графички приказ овог модула у развојном окружењу.



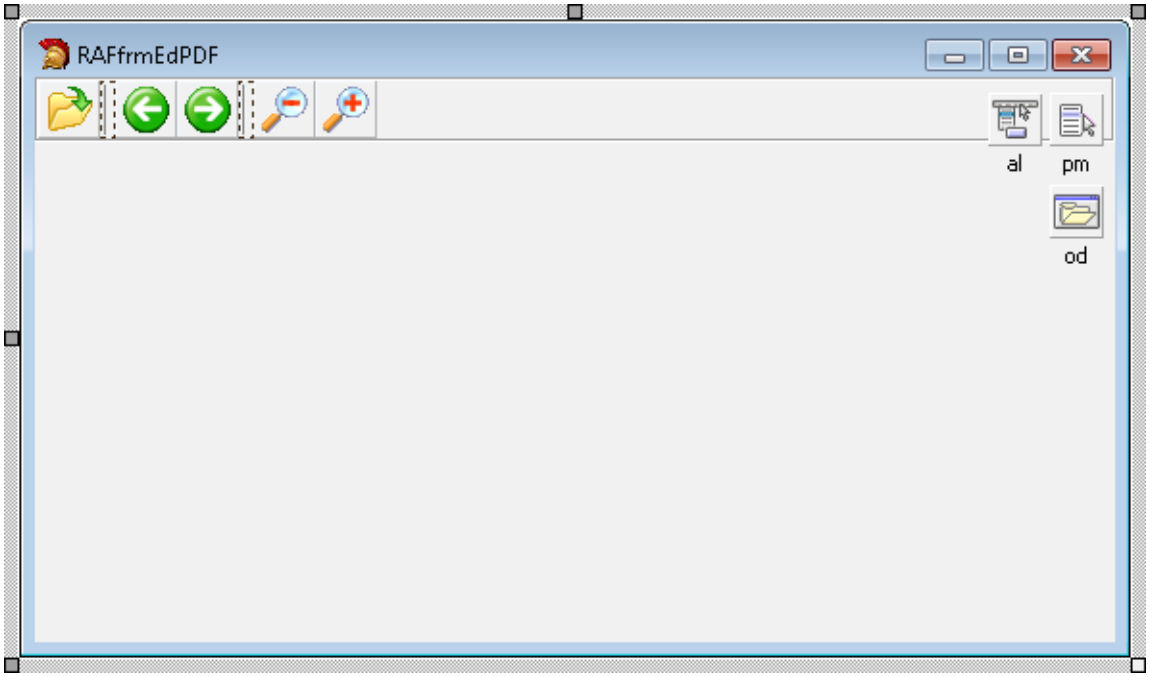
Слика 38 - URAFfrmEdPresentersScreen.pas

Снимак екрана се приказује помоћу компоненте типа **TVLDSImageDisplay** из пакета *Митов Видео Лабс* (члан **imScreen**).

Компонента **imScreen** је својством **InputPin** повезана са својством **OutputPin** компоненте **ScreenOutput** која припада објекту типа **TRAFdmAVSStreamDecoder** који обрађује податке које шаље предавач.

Када клијент ради у режиму слушаоца предавања, ово је једини подпрозор који може да се користи. Сви други подпрозори могу да се користе само када клијент ради у режиму предавача.

URAFfrmEdPDF.pas дефинише класу **TRAFfrmEdPDF** која имплементира подпрозор за приказ ПДФ докумената. На слици је графички приказ овог модула у развојном окружењу.

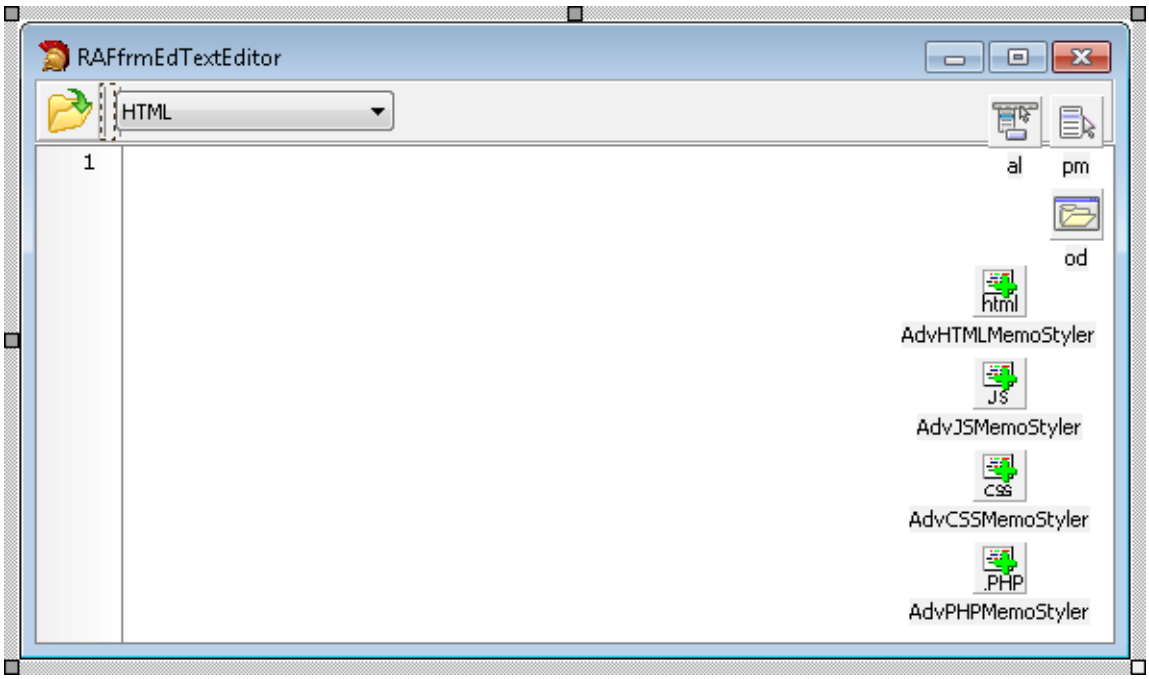


Слика 39 - URAFfrmEdPDF.pas

ПДФ документ се приказује помоћу *Актив Икс* компоненте *Акро ПДФ* из пакета *Адоби Акробат Браузер Тајп Лајбрари* (члан **FAcroPDF**). Компонента **od** је типа **TOpenDialog** и имплементира стандардни дијалог за отварање датотека у оперативном систему *Мајкрософт Виндоуз*.

Имплементиране су основне функционалности за отварање, навигацију и скалирање ПДФ документа.

URAFfrmEdTextEditor.pas дефинише класу **TRAFfrmEdTextEditor** која имплементира подпрозор за приказ текстуалних докумената са бојењем текста. На слици је графички приказ овог модула у развојном окружењу.

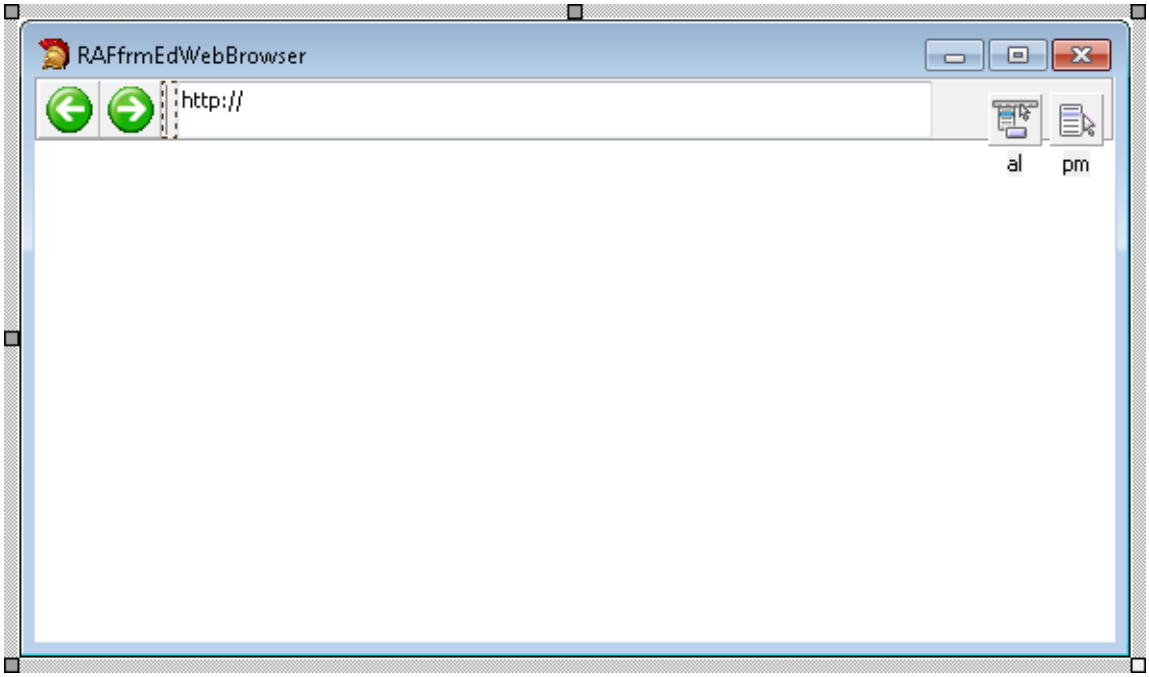


Слика 40 - URAFfrmEdTextEditor.pas

За приказ текстуалног фајла се користи компонента *Адвансд Мемо* из пакета *TMC Компонент Пек* (члан **`AdvMemo`**). За бојење текста се користе додаци за *Адвансд Мемо*, и то за језике ХТМЛ, Јаваскрипт, ЦСС и ПХП (чланови **`AdvHTMLMemoStyler`, `AdvJSMemoStyler`, `AdvCSSMemoStyler`, `AdvPHPMemoStyler`**).

Имплементиране су основне функционалности за отварање и навигацију кроз текстуални документ и избор стила који ће се користити за бојење текста.

`URAFfrmEdWebBrowser.pas` дефинише класу **`TRAFfrmEdWebBrowser`** која имплементира подпрозор за приказ веб садржаја (веб читач). На слици је графички приказ овог модула у развојном окружењу.

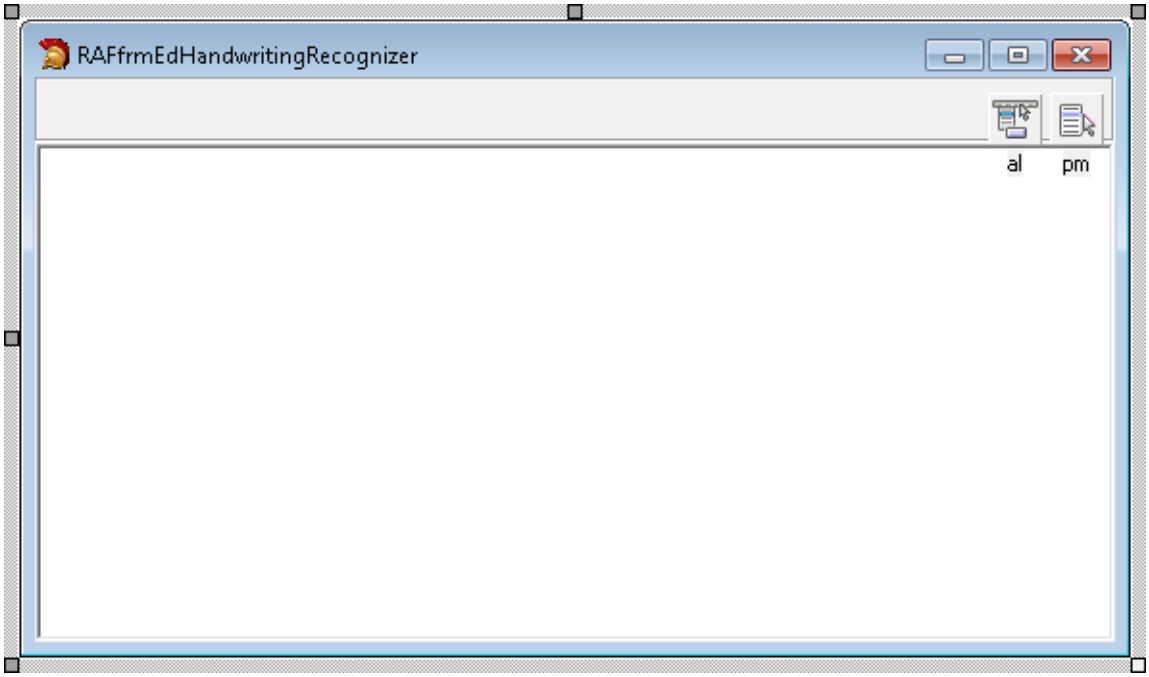


Слика 41 - URAFfrmEdWebBrowser.pas

За приказ веб садржаја се користи компонента *Веб Браузер*, из стандардне библиотеке визуелних компоненти (члан **`WebBrowser`**).

Имплементиране су основне функционалности за навигацију кроз веб садржаје.

`URAFfrmEdHandwritingRecognizer.pas` дефинише класу **`TRAFfrmEdHandwritingRecognizer`** која имплементира подпрозор са таблом за писање са препознавањем рукописа. На слици је графички приказ овог модула у развојном окружењу.

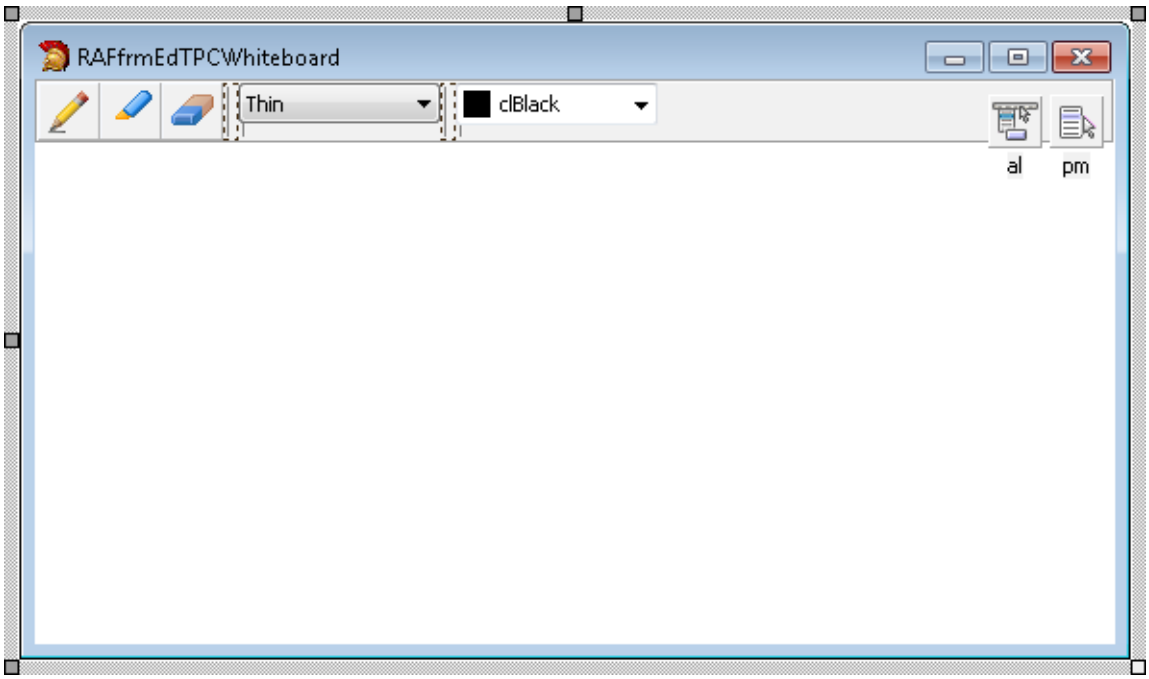


Слика 42 - URAFfrmEdHandwritingRecognizer.pas

За препознавање рукописа се користи компонента *Инк Едит*, из пакета *Мајкрософт Виндоуз Икс Пе Таблет Пи Си Едишн Рекогнајзер Пек* (члан **`InkEdit`**).

Овај подпрозор је имплементиран првенствено као пример да се употребом софтверске компоненте може постићи веома напредна функционалности без и једне линије писаног кода. Сав изворни код у овом модулу је аутоматски генерисан од стране развојног окружења.

`URAFfrmEdTPCWhiteboard.pas` дефинише класу **`TRAFfrmEdTPCWhiteboard`** која имплементира подпрозор са таблом за цртање и писање. На слици је графички приказ овог модула у развојном окружењу.



Слика 43 - URAFfrmEdTPCWhiteboard.pas

За реализацију табле за цртање и писање се користи компонента *Инк Пикчр*, из пакета *Мајкрософт Таблет Пи Си Тајп Лајбрари* (члан **`InkPicture`**).

Имплементиране су основне функционалности за цртање и писање по табли, попут избора оловке или маркера, брисања последње линије и избора дебљине и боје линије.

По покретању програма, приказују се главни прозор и дијалог за унос корисничког имена и лозинке. Након повезивања са сервером и успешне провере идентитета, корисник може издати команду којом прави групу и узима улогу предавача, или команду којом се придружује предавању. Ако клијент ради у режиму предавача, он може отворити било који подпрозор осим **`TRAFfrmEdPresenterScreen`** и помоћу њих приказивати наставни материјал. Ако клијент ради у режиму слушаоца, он може отворити само подпрозор **`TRAFfrmEdPresenterScreen`** преко кога ће пратити наставни материјал који приказује предавач. Када пожели, учесник може напустити групу и одјавити се са система.

Из описа имплементације система се види да је у потпуности испуњен план употребе софтверских компоненти, чиме је значајно убрзан развој. Анализом изворног кода, види се да је највећи део изворног кода аутоматски генерисан од стране развојног окружења. Број писаних линија кода је изузетно мали у односу на функционалности које систем садржи. Количина писаног кода је у највећем броју модула испод 20% од укупног броја линија кода, а у неким модулима га уопште нема (препознавање рукописа).

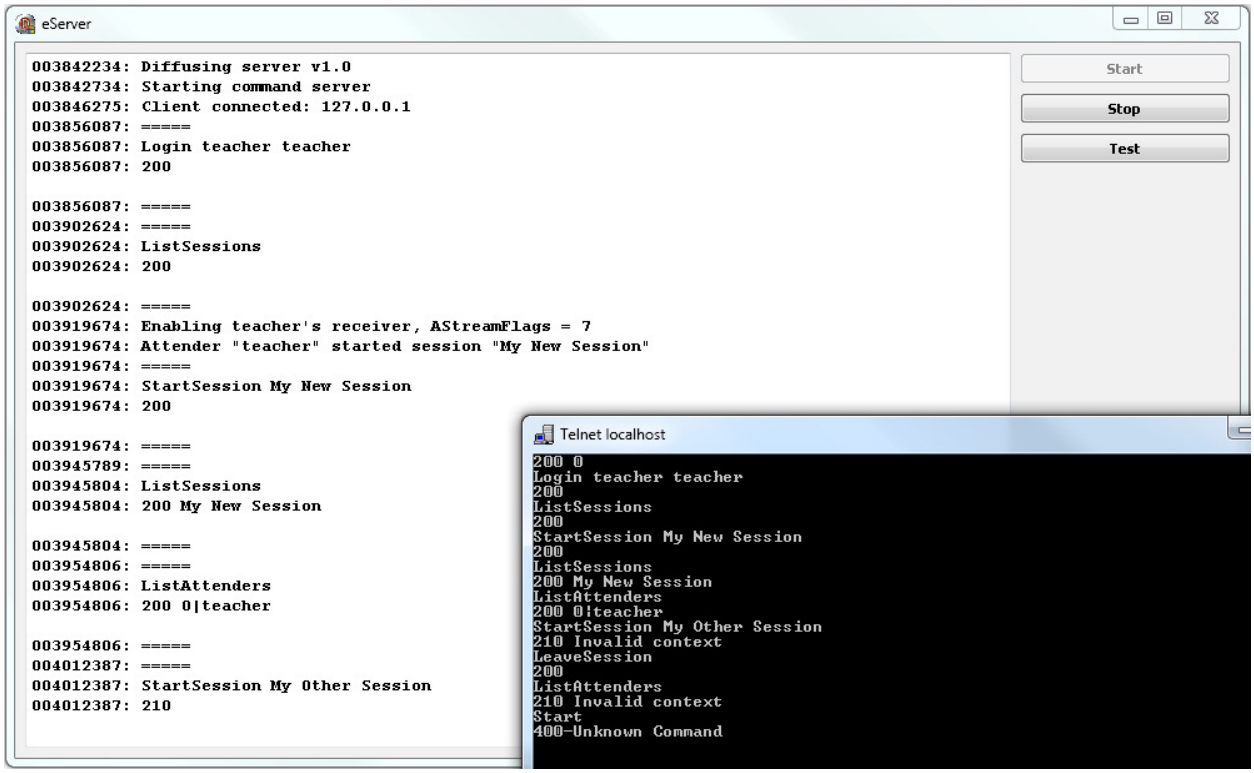
3.4 Тестирање и демонстрација

Писање програма у првом циклусу израде прототипа је завршено. Задовољни смо што смо интензивном употребом модела брзог развоја, уз мала улагања у времену и новцу, елегантно покрили постављене захтеве. Сада постављамо питање исправности имплементације.

За потребе тестирања исправности, изворни код је допуњен многим дијагностичким порукама које ће се исписивати током извршавања програма.

3.4.1 Тестирање исправности имплементације сервера

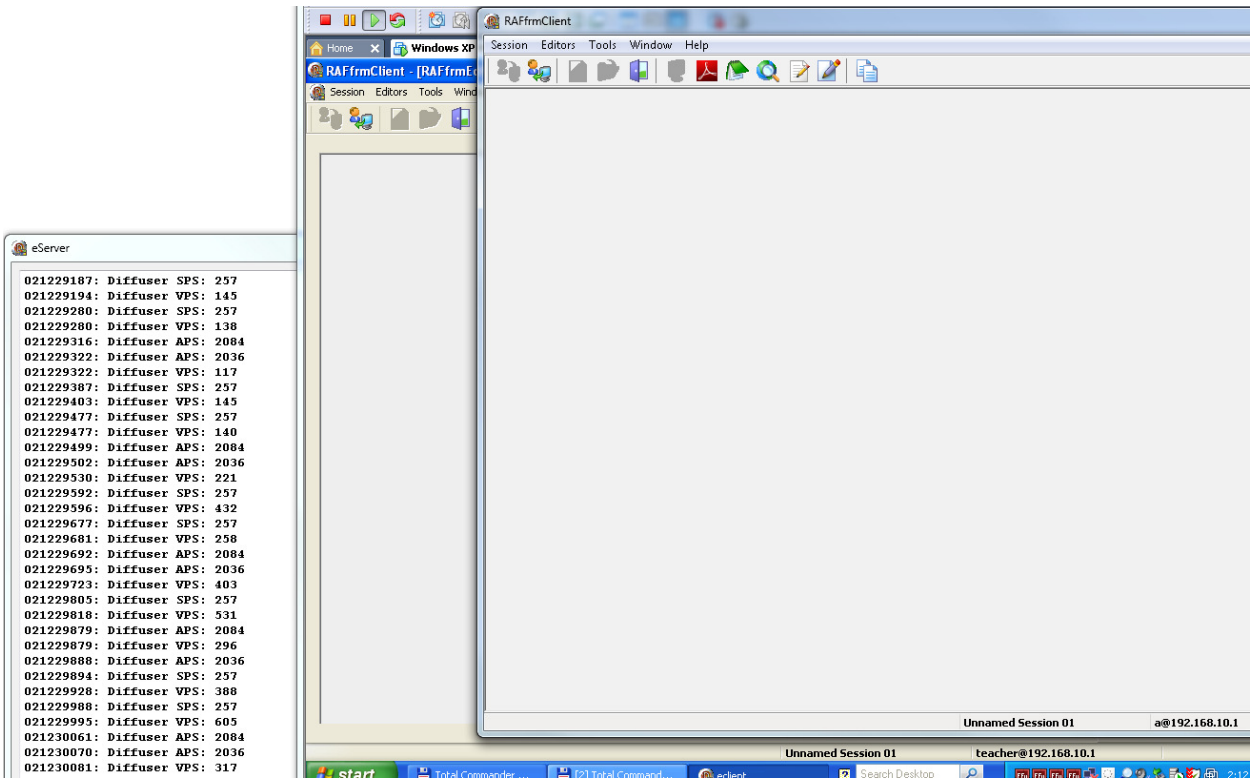
Тестирање исправности имплементације сервера почињемо од командног протокола. Покрећемо серверски део система, и притиском на дугме *Тест* покрећемо телнет сесију са командним сервером. За свако стање коначног аутомата (има их 4, а у 3 се могу издавати команде) издајемо све команде из табеле. Пратимо дијагностичке поруке које сервер исписује. Треба да се уверимо да правила издавања команди прате коначни аутомат и да се у складу са задатим командама објекти у подсистему за управљање групама учесника исправно мењају, а ТЦП и УДП сервери исправно активирају и деактивирају.



Слика 44 - Детаљ током тестирања серверске стране командног протокола

Дијагностичке поруке из сервера нам показују да правила издавања команди заиста прате коначни аутомат, да се у складу са задатим командама објекти у подсистему за управљање групама учесника исправно мењају, и да се ТЦП и УДП сервери исправно активирају и деактивирају, па је закључак да је овај део система исправно имплементиран.

Други тест сервера треба да покаже да сервер исправно прима и прослеђује податке. За ово ће нам бити потребна два клијента. Једног клијент ће се повезати на сервер и покренути предавање, а други ће се придружити предавању. Дијагностичке поруке сервера треба да покажу да сервер прима и прослеђује податке.



Слика 45 - Детаљ током тестирања подсистема за пријем и прослеђивање података

На слици се виде два клијента, један ради на истом рачунару на коме ради сервер, а други ради у виртуелној машини. Оба клијента су повезана на сервер, а клијент који је на слици у првом плану ради у режиму предавача. Дијагностичке поруке сервера показују да сервер уредно прима податке преко сва три комуникациона канала, тј. аудио (**APS**) и видео (**VPS**) податке и снимак екрана (**SPS**). Метода из које се исписује дијагностичка порука се позива када сервер прими податке, и из исте методе се подаци прослеђују другим клијентима. Из овога закључујемо да је имплементација подсистема за пријем и прослеђивање података исправна.

Из претходна два теста закључујемо да је имплементација сервера исправна.

3.4.2 Тестирање исправности имплементације клијента

Тестирање исправности имплементације клијента почињемо од корисничког интерфејса.

Прво ћемо задавати разне команде и пратићемо да ли се кориснички интерфејс прилагођава контексту издавања команди, тј. да ли искључује елементе корисничког интерфејса преко којих се позивају команде или делови програма који се не могу користити у текућем контексту.

На почетку рада програма, укључене су само команде за пријаву на систем и излаз из програма. По пријави на систем, искључује се команда за пријаву на систем, а укључују се команда за одјаву са система и команде за покретање или придруживање предавању. По покретању предавања, искључују се команде за покретање или придруживање предавању, а укључују се команде преко којих се позивају подпрозори за представљање наставног материјала. По напуштању предавања, искључују се команде преко којих се позивају подпрозори за представљање наставног материјала, а укључују команде за покретање или придруживање предавању. По придруживању предавању, искључују се команде за покретање или придруживање предавању, а укључује команда за позивање подпрозора који приказује снимак екрана предавача. По одјављивању са система, искључују се све

команде, а укључује се команда за пријаву на систем. Овакво понашање је у потпуном складу са правилима коначног аутомата, па закључујемо да је имплементација прилагођавања корисничког интерфејса исправна.

Други део теста корисничког интерфејса проверава да ли подпрозори за приказ наставног материјала раде исправно. Провераваћемо функционалности сваког од њих посебно, задавањем команди које садрже.

Подпрозор за приказивање снимка екрана ћемо тестирати када будемо тестирали делове за генерисање и пренос података.

Подпрозор за приказивање ПДФ докумената успешно позива дијалог за отварање фајлова, али дијалог не филтрира фајлове по екстензији. Изабрани ПДФ документ се успешно учитава и приказује. Навигација и скалирање раде исправно.

Подпрозор за приказивање текстуалних докумената са бојењем текста успешно позива дијалог за отварање фајлова, али дијалог такође не филтрира фајлове по екстензији. Изабрани текстуални документ се успешно учитава и приказује. Бојење синтаксе ради исправно.

Подпрозор за приказивање веб садржаја успешно учитава задату веб адресу. Навигација по страници ради исправно. Навигациони дугмићи са стрелицама раде исправно.

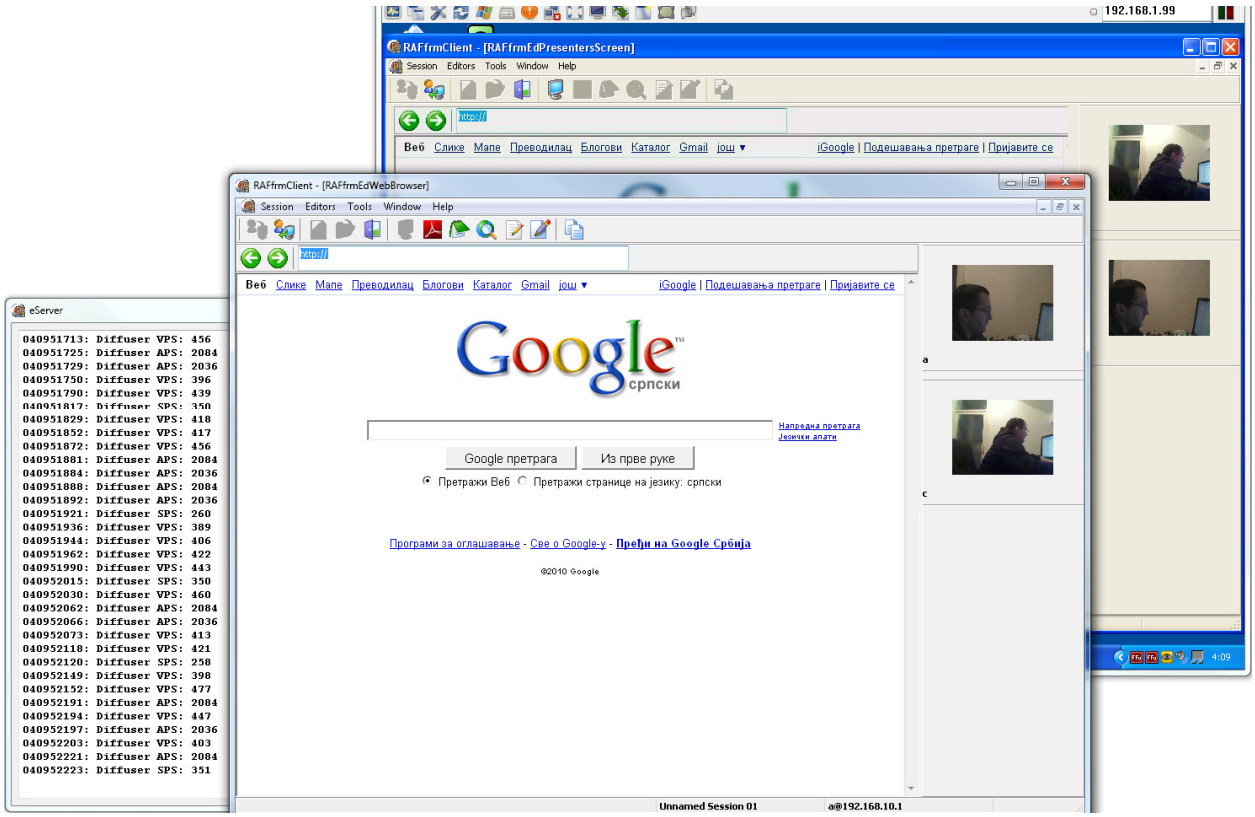
Подпрозор са таблом за писање са препознавањем рукописа исправно реагује на покрете миша и оловке са графичке табле, препознаје бројеве, симболе и латинична слова енглеске абецеде, али је потребно писати доста читко. Испоставља се да је језик препознавања рукописа подешавање на нивоу оперативног система.

Подпрозор са таблом за цртање и писање исправно реагује на покрете миша и оловке са графичке табле. Режими оловке и маркера раде исправно. Избори дебљине и боје линије раде исправно.

Копирање садржаја активног подпрозора на таблу за цртање и писање ради исправно, с тим да се понекад постављена позадина не исцрта у потпуности.

Из претходног закључујемо да је имплементација графичког корисничког интерфејса исправна, са изузетком прозора за приказ снимка екрана који још није тестиран.

Следећи тест се бави деловима програма за генерисање, слање и пријем података. За ово ће нам, слично као код тестирања преноса података на серверу, бити потребна два клијента. Једног клијент ће се повезати на сервер и покренути предавање, а други ће се придружити предавању. Клијенти треба да виде и чују један другог, а клијент који ради у режиму слушаоца треба да види снимак екрана предавача.



Слика 46 - Детаљ током тестирања подсистема за генерисање, пренос и пријем података

На слици се виде два клијента, један ради на истом рачунару на коме ради сервер, а други ради на другом рачунару чију радну површину посматрамо помоћу програма за приступ са удаљене локације. Оба клијента су повезана на сервер, а клијент који је на слици у првом плану ради у режиму предавача. Дијагностичке поруке сервера показују да сервер уредно прима податке преко сва три комуникациона канала, тј. аудио (**APS**) и видео (**VPS**) податке и снимак екрана (**SPS**). Клијенти виде и чују један другог, а клијент који ради у режиму слушаоца, може да види снимак екрана предавача. Ово нам уједно и показује да је имплементација прозора за приказ снимка екрана предавача исправна. Дешава се да дође до микрофоније уколико је јачина звука на звучницима нешто већа, али ово може бити последица тестирања у брзој локалној мрежи и непосредне близине клијентских рачунара. Ипак, појава микрофоније указује да у реалној ситуацији може доћи до ехоа. Овим једноставним тестом смо потврдили да подсистем за генерисање, пренос и пријем података ради исправно.

Из претходних тестова закључујемо да је имплементација клијента исправна.

Из тестова сервера и клијента закључујемо да систем задовољава минимум функционалности, да може да ради исправно и да је спреман за демонстрацију.

3.4.3 Демонстрација

За потребе демонстрације систем је инсталиран и пуштен у рад у окружењу за демонстрацију које је слично реалном радном окружењу. Окружење се демонстрацију се састоји од 5 рачунара новије генерације, повезаних у мрежу преко интерфејса брзине 100Mbps. Током демонстрације анализирамо још нека важна својства извршавања система и уочавамо проблеме у раду.

Серверски програм се брзо покреће. У неактивном стању, тј. када нема активних предавања, користи око 1,5 MB радне меморије, има 5 нити извршавања и користи

занемарљиво мало процесорског времена. У потпуно активном стању, током трајања предавања, употреба радне меморије се повећава на око 2,5 MB и за сваког клијента још по око 200 kB, број нити извршавања се пење на 8 и за сваког клијента још по 2, а употреба процесорског времена се незнатно повећава за сваког новог клијента. Додатне 3 нити потичу од активирања УДП сервера, а по 2 за сваког клијента потичу од повезивања на командни сервер. Иако имамо потенцијално велики број нити, све нити везане за командни сервер су углавном неактивне и само се спорадично активирају, тако да велики број нити неће оптерећивати систем. Релативно ниска захтевност серверског програма није изненађење, имајући у виду његову једноставну имплементацију.

Клијентски програм се брзо покреће. У неактивном стању, тј. када је повезан са сервером, али нема активног предавања, користи око 10 MB радне меморије, има 9 нити извршавања, и троши занемарљиво мало процесорског времена. У активном стању, у режиму предавача, без слушалаца, употреба радне меморије се повећава на око 50 MB, број нити извршавања се пење на 31, а употреба процесорског времена је у границама 10-20%. У активном стању, у режиму слушаоца, само са предавачем, употреба радне меморије се повећава на око 35 MB, број нити извршавања се пење на 58, а употреба процесорског времена је у границама 10-20%. За сваког новог учесника, употреба радне меморије се повећава за око 5 MB, број нити расте за око 10, а употреба процесорског времена за 3-5%. Разлог за велики пораст захтевности је изабрана технологија за прикупљање и обраду аудио и видео података и снимка екрана (*Дајрект Шоу*).

Мрежни саобраћај, за једног клијента, по комуникациним каналима, је следећих редова величина: аудио 160 kbps, видео 12-80 kbps и снимак екрана 16kbps-1,2Mbps. Снимак екрана захтева велики проток када слика садржи велике промене у односу на претходни фрејм.

Долази до ехоа у аудио сигналима.

Уколико дође до губитка фрејма снимка екрана, слика се деформише.

Подрозори који користе *Актив Икс* компоненте се значајно спорије отварају од подпрозора који их не користе.

Клијентски програм пријављује грешку ако се отвара подпрозор који користи *Актив Икс* компоненту која није инсталирана на систему. Ово је критичан проблем, јер значајно омета употребу програма.

Дешава се да *Актив Икс* компонента *Акро ПДФ*, из подпрозора за приказ ПДФ докумената, непредвидљиво откаже без поруке о грешки. Проблем не изненађује, јер првенствена намена ове компоненте није употреба као софтверске компоненте за развој софтвера, већ подршка за приказ ПДФ докумената у веб читачима. Због тога произвођач и не даје подршку за компоненту као развојни алат. Ово је критичан проблем, јер значајно омета употребу програма.

3.5 Процена и измена захтева и дизајна

Имајући у виду резултате тестирања и запажања са демонстрације, изводимо закључке о потребним изменама у захтевима и дизајну система и програма.

У ситуацији када у предавању учествују само предавач и један слушалац, пренос података уз посредовање сервера је непотребно оптерећење. Треба омогућити ад-хок повезивање у коме ће два клијента након провере идентитета директно размењивати податке, без посредовања сервера.

За потребе покривања великог броја клијената, у серверу треба имплементирати режим репетитора тј. рефлектора саобраћаја, у коме ће се сервер А повезивати на сервер Б, и прослеђивати му саобраћај од свих својих клијената, а од сервера Б примати саобраћај од свих његових клијената. На овај начин се оптерећење прослеђивања података може распоредити на више рачунара.

Клијентски програм је превише захтеван, у смислу рачунарских ресурса, и треба пронаћи решење за растерећивање. Главни узрок је изабрана технологија за прикупљање и обраду аудио и видео података и снимка екрана, и то у декодерима, јер се за сваког клијента покреће и до 20 нових нити извршавања.

Аудио канал се може унапредити тако што ће се у зависности од нивоа аудио сигнала, покретати или заустављати пренос аудио података. Другим речима, ако на аудио каналу има корисних података, тј. ако учесник говори, његови аудио подаци се шаљу серверу, а ако учесник ћути, његови аудио подаци се не шаљу серверу. Ово ће значајно смањити количину аудио података који се преносе, јер ће највећи део времена сви осим једног учесника ћутати.

Аудио канал се може унапредити и тако што ће се сви пристигли аудио подаци на серверу миксовати у један сигнал, и онда ће се тај сигнал слати клијентима. Овакав приступ ће унети извесно кашњење у аудио сигнал и оптеретити сервер, али ће растеретити аудио декодер и за било који број учесника ће бити довољан само један аудио декодер. Ово је врло значајно унапређење.

Могућност да се сви учесници у сваком тренутку међусобно виде је корисна, али је упитно колико је то заиста значајно за предавање, обзиром да је најважније видети онога ко тренутно говори. Видео канал се може унапредити слично као и аудио канал. У зависности од нивоа аудио сигнала ће се покретати или заустављати и пренос видео података. Ако корисник говори, његови видео подаци се шаљу серверу, јер је важно видети онога ко говори. Слично као код аудио података, ово би значајно смањило количину видео података који се преносе, јер ће највећи део времена, сви осим једног учесника ћутати.

Такође, сви пристигли видео подаци се могу на серверу миксовати у један низ слика, и онда би се само тај низ слика слао клијентима. Опет, слично као и код аудио података, овај приступ ће унети извесно кашњење у видео сигнал и оптеретити сервер, али ће растеретити видео декодер и за било који број учесника ће бити довољан само један видео декодер.

Проблематичност овог приступа лежи у томе да се сервер може значајно оптеретити обрадом аудио и видео података, па је потребно наћи компромис између растерећења клијента и оптерећења сервера.

Прикупљање и обрада снимка екрана се такође може ефикасније имплементирати. За ефикасније прикупљање снимка екрана се може користити *ВНЦ Мирор Драјвер* (VNC Mirror Driver), који се користи у програму *Ултра ВНЦ* (Ultr@VNC). Овај драјвер има директан приступ видео меморији и најбрже могуће добија обавештења о променама на екрану. Додатно, драјвер има имплементирану могућност да аутоматски проналази и прослеђује позиваоцу само измењене делове екрана, што је од изузетне важности за ефикасан пренос снимка екрана преко рачунарске мреже. Драјвер је слободно доступан на веб страници произвођача.

Снимак екрана се може кодирати и компресовати специјалним алгоритмом прилагођеним за снимке екрана. Кодек који имплементира такав алгоритам је *Тексмит Скрин Кепчр Кодек* (TechSmith Screen Capture Codec). *Тексмит* је произвођач програма *Камтазија* (Camtasia), познатог програма за снимање садржаја екрана. Кодек је слободно доступан на веб страници произвођача.

За аудио сигнале је потребно имплементирати и поништавање ехоа.

Клијентски програм је зависган од *Актив Икс* компоненти, па се у инсталациони пакет програма морају укључити све *Актив Икс* компоненте које програм захтева.

Клијентски програм је зависган од кодека за аудио и видео податке и снимке екрана. У инсталациони пакет програма се морају укључити сви кодеци које програм захтева.

Због тога што отказује без поруке о грешки и што није намењена за развој, посебно је проблематична компонента *Акро ПДФ*. Уместо ове компоненте, за приказ ПДФ докумената се могу користити компонента из пакета *ПДФ Тулкит ВЦЛ* (PDF Toolkit VCL), компаније *Гностис* (Gnostice). *ПДФ Тулкит ВЦЛ* је комерцијална развојна библиотека за *РАД Студио*. Иако компонента *Акро ПДФ* нуди најбољу компатибилност са ПДФ документима, њено непредвидљиво отказивање је исувише велики проблем да би се игнорисао.

Клијентски програм не нуди никакве могућности подешавања. Треба имплементирати подешавања попут избора аудио и видео извора, тј. избора микрофона и камере, подешавања јачине звука са микрофона, подешавање јачина звука репродукције и слично.

У клијентском програму се може имплементирати и снимање предавања у датотеку. Ово се може веома лако постићи, употребом додатних компоненти из пакета Митов Видео Лаб.

Кориснички интерфејс клијента може бити модернији, може се имплементирати популарни *Офис 2007 Рибон* (Office 2007 Ribbon) интерфејс.

Тестови у другом циклусу ће такође бити усмерени само на испитивање да ли је исправност имплементације функционалности довољна за демонстрацију.

За коначни прелазак у фазу писања програма у другом циклусу развоја, предложеним изменама се могу придружити приоритети и унапред одредити које од измена ће бити изведене у другом, а које у наредним циклусима.

4 Литература

1. Софтверско инжењерство: Теорија и пракса – Шари Лоренс Флигер и Џоана Атли ЦЕТ
   http://www.cet.rs/cetknjige/KDetaljno.aspx?ID=3440
2. Steve McConnell - Rapid Devlopment: Taming Wild Sotware Scheduless Microsoft Press
   http://www.amazon.com/Rapid-Development-Taming-Software-Schedules/dp/1556159005/
3. http://en.wikipedia.org/wiki/Software_development_methodology
4. http://en.wikipedia.org/wiki/Software_development_process
5. http://www.jamesmartin.com/
6. http://www.blueink.biz/RapidApplicationDevelopment.aspx
7. http://www.cs.bgsu.edu/maner/domains/RAD.htm
8. Practical UML: A Hands-On Introduction for Developers
   http://edn.embarcadero.com/article/31863
9. Object Pascal Style Guide
   http://edn.embarcadero.com/article/10280