****

**ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ УНИВЕРЗИТЕТА У БЕОГРАДУ**

*КАТЕДРА ЗА РАЧУНАРСКУ ТЕХНИКУ И ИНФОРМАТИКУ*

**ДОМАЋИ ИЗ КОНКУРЕНТНОГ И ДИСТРИБУИРАНОГ ПРОГРАМИРАЊА (ЈАВА ЛИНДА – RMI)**

Студент : *Страхиња Јањић* Индекс : *0411/2015*

Датум прве верзије: *29.01.2020* Датум финалне верзије*: 02.03.2020*

Предметни наставник: Место одбране:

*Захарије Радивојевић* Лабораторија 25 Павиљон Рашовић

Садржај

*1. Текст Задатак1*

*2. Приступ решавању проблема2*

*3. Упутство за коришћење софтвера3*

*4. Функционална спецификација система3*

*5. Недостаци приложеног решења6*

*6. Закључак6*

1. **Текст задатка:**
   * + - 1. 1. Централни сервер који служи за праћење рада извршавања дистрибуиране обраде, чување информација о доступним чворовима у мрежи и могућност поновног стартовања појединих послова.

2. Радна станица која од централног сервера добија послове које треба одрадити.

3. Кориснички програм, који задаје посао који треба урадити и његове параметре.

Пројектовати дистрибуирани рачунарски систем који треба да омогући дистрибуирану обраду и међусобну удаљену синхронизацију временски захтевних послова. Ова синхронизација треба да се обави по угледу на виртуелни заједнички простор торки заједнички за све послове унутар скупа послова који постоји код библиотеке *C-Linda*.

Програм треба да ради у систему који се састоји од више рачунара повезаних у LAN (*Local Area Network*) или WAN (*Wide Area Network*).

У систему постоји три типа програма:

Процес започиње тако што кориснички програм задаје посао који је потребно обрадити. Овај посао представља програм писан у програмском језику Јава запакован у *jar* архиву. Након тога кориснички програм контактира централни сервер коме прослеђује тај посао и параметре потребне за његову обраду. Када централни сервер прими посао и његове параметре, он га прослеђује једној радној станици, чека резултат обраде и враћа комплетан резултат клијенту.

Када радна станица прими посао започиње са његовим извршавањем, на основу примљених параметара. Посао се извршава на радној станици тако што радна станица покреће посао који је дат као *jar* архива којој прослеђује параметре које је клијент задао. Као посебан параметар поставља путању до *jar* архиве у којој се налази библиотека за удаљену синхронизацију. Та библиотека је реализована по угледу на библиотеку *C-Linda* која се користи за синхронизацију. Интерфејс који је потребно за задовољи је дат у прилогу овог документа.

Пошто се у датом корисничком послу може покренути више послова (метода **eval**(...)), они треба да буду распоређени на слободне радне станице. Када се заврши извршавање програма, радна станица прикупљене излазне токове података прослеђује серверу, као и резултате извршавања. Да би се обезбедило прикупљање информација о активним радним станицама централни сервер на сваких ***x*** секунди проверава да ли је нека радна станица исправна. Уколико није исправна, обавештава корисника који је посао престао да се извршава. Након тога корисник треба да одлучи да ли посао прекинути или да се прекинути део проследи некој слободној радној станици. Уколико корисник није доступан, прекида се извршавање целог посла.

Након слања захтева за обраду кориснички програм може да раскине везу са централним сервером. Веза може бити раскинута гашењем програма или затварањем комуникационог канала. Када се следећи пут повеже, кориснички програм може да тражи резултате претходно задате обраде. Треба обезбедити да централни сервер може да у паралели да прима већи број послова које је потребно обрадити. Кориснички програм може од централног сервера да тражи информације о статусу посла, а може да тражи и резултате. Одмах по стартовању, радне станице шаљу централном серверу информацију о томе да су стартоване и број послова које могу у паралели да обрађују.

Параметри посла које клијент задаје су: команда која се задаје Јава виртуелној машини да би покренуо архиву, датотеке које са клијентског рачунара треба пребацити на радну станицу да би команде могле да се изврше (не више од 6), датотеке које са радне станице треба пребацити на клијентски рачунар да које представљају резултате (не више од 6). Ови параметри могу да се задају или путем корисничког интерфејса или путем текстуалне датотеке.

Централни сервер у лог уписује време када је пристигао сваки посао, број под којим је посао сачуван, име рачунара коме је посао прослеђен, време када је посао завршен и његов тренутни статус. Статус посла може да буде: *Ready* – приспео на сервер, али није никоме прослеђен, *Scheduled* – тренутно се прослеђује радној станици, *Running* – извршавање је у току, *Done* – посао се успешно извршио, *Failed* – посао није могао да се изврши, *Aborted* – корисник је одустао од извршавања посла.

Проблем решити искључиво користећи **Јава RMI** мрежну комуникацију. Решење треба да буде независно од посла који се обавља. За сваки од ова три типа рачунара треба да постоји одговарајући графички кориснички интерфејс (GUI треба да буде развијен користећи Java **SWING** компоненте). Радна станица треба да има могућност покретања и без корисничког интерфејса.

1. **Приступ решавању проблема:**

Решавање датог проблема је вишеслојно и захтева добро познавање готово свих нивоа апстракције у програмирању, поготово добро познавање Јава Технологија (RMI, SWING) као и јава виртуелног окружења.

Први корак реализације био је свакако писање библиотеке Линда која је првенствено требало да ради конкурентно па потом и дистрибуирано. Од предметних наставника достављена је једна могућа реализација дате библиотеке названа LocalLinda. Главни проблем са овом библиотеком био је сто није била погодна за дистрибуирани рад, јер су коришћени концепти за синхронизацију на нивоу нити, а не на нивоу процеса. Такође интерфејс саме Линда библиотеке је дат тако да су параметри торки прослеђивани у облику стринга сто није бас најфлексибилније решење. Због свега наведеног аутор овог рада се одлучио за приступ где ће се направити класа торки (***public*** ***class*** Tuple ***extends*** LinkedList<Serializable> ***implements*** Serializable), чија функција би била да енкапсулира торке достављене у облику стринга и да их сачува у њиховом правом формату, осигуравајући да елементи торки имплементирају интерефејс Serializable. Ово је делом урађено како би нам касније омогућило елегантан начин за дистрибуирану синхронизацију блокирајућих позива Линда библиотеке. Ова наша клас торки је део “util” пакета и њене методе могу да баце изузетак (TupleFormatException) у случају да неки од прослеђених параметара није коректног облика.

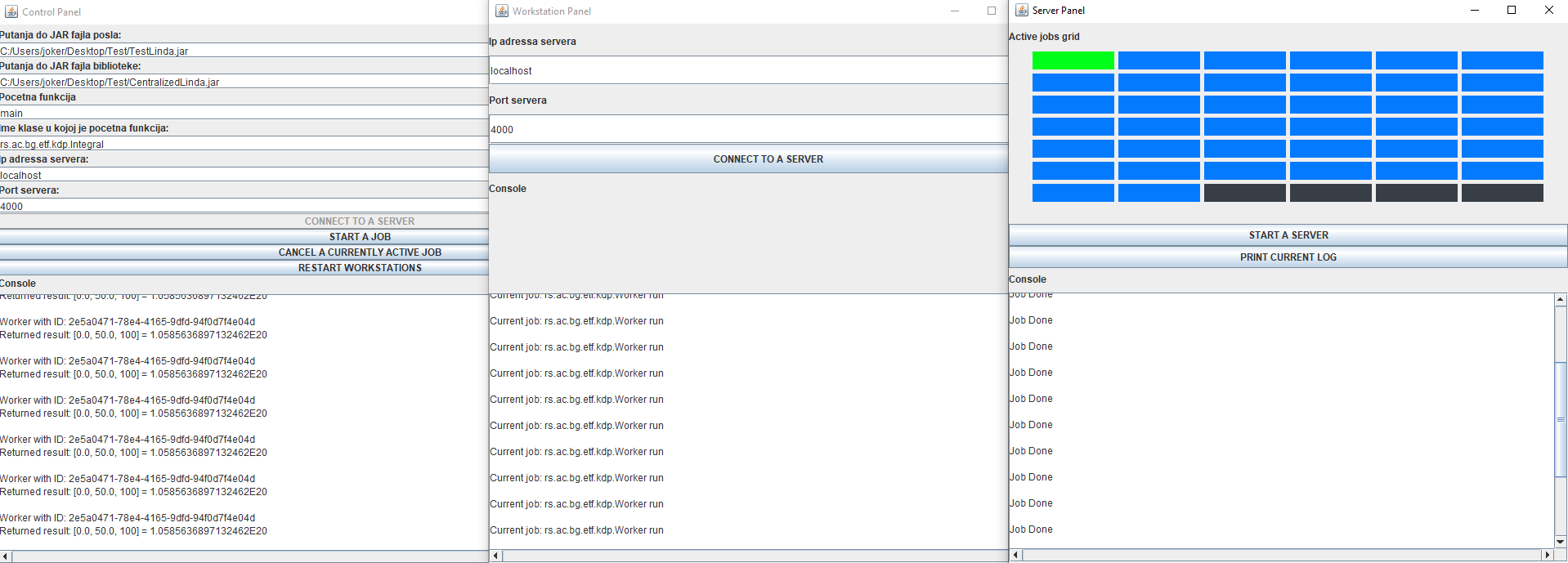
Следећу класу коју смо дефинисали је класа која имплементира саму библиотеку Линда (***public*** ***class*** CentralizedLinda ***implements*** Linda), ова класа пружа могућност за удаљену синхронизацију. Имплементирана је на такав начин да поседује простор торки који је суштини уланчана листа горе дефинисане класе Tuple, у ком се чувају све торке. Систем за унутрашњу синхринизацију саме бибилиотеке користе се Reentrant Lock и Condition из јавиног concurrency пакета. Зато што смо дефинисали класу за торке на такав начин да чува објекте а не стрингове сада имамо могућност да креирамо два засебна простора који ће чувати блокирајуће захтеве за читање и дохватање елемента. При доласку тог захтева а ми немамо ту торку у простору торки, ми ту торку стављамо у одговарајући простор заједно са условом на коме се захтев блокирао. При додавању торке у простор торки пи пролазимо кроз оба простора и дохватамо један услов и сигнализирамо да један корисник може да прочита или узме торку из простора торки. Овим системом добили смо поуздано и дистрибуирано решење проблеме библиотеке за удаљену синхронизацију.

Пошто сада имам стабилну и библиотеку за удаљену синхронизацију прелазимо на реализацију серверског дела. Овде смо прво направили интерфејс (***public*** ***interface*** LindaRMI ***extends*** Remote) за наш Линда сервер који мора да има имплементиране све методе Линда библиотеке како би удаљени позиви ка том серверу били енкапсулирани на клијентској страни. Наш Линда сервер имплементира дати интерфејс на такав начин да прави једну инстанцу наше библиотеке и позива њене методе.

Линда клијети су LindaManager и LindaWorkstation они имплементирјау интерфејс (Linda) на такав начин да њихове методе зову серверске методе удаљеним позивом.

Све три главне активне компоненте система имају графички кориснички интерфејс реализован у SWING графичкој библиотеци и њихови интерфејси изгледају овако :

Слика 1. *Пример изгледа графичког интерфејса*



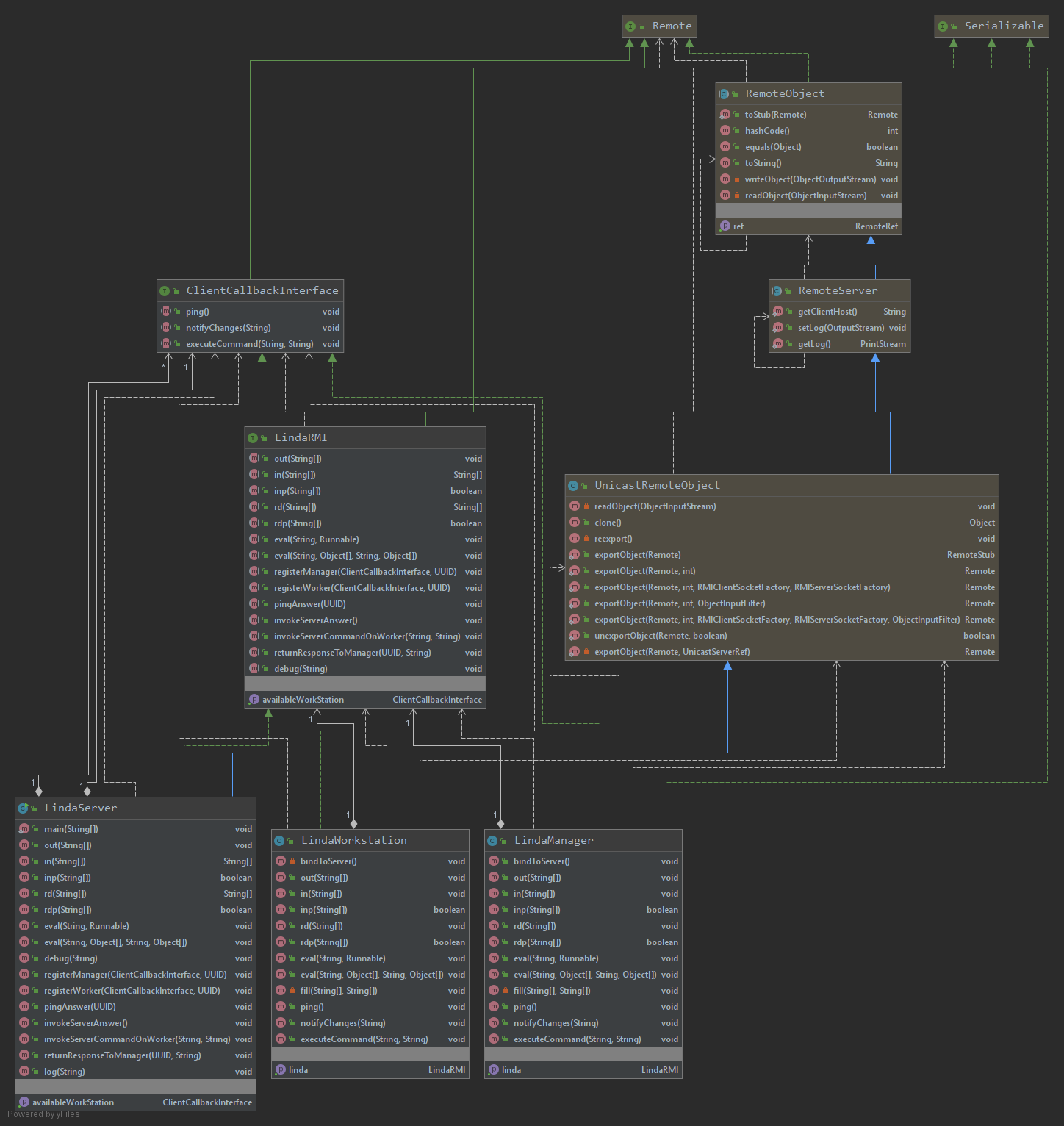
1. **Упутство за коришћење софтвера:**

Софтвер се састоји од три апликације које могу независно да се покрећу на више рачунара, те апликације су Радна станица (Workstation), Сервер (Server), Кориснички програм (Control Panel). Да би систем у целини могао да ради првенствено мора да се покрене серверска апликација и да се притисне дугме (START A SERVER), зато што све друге компоненте комуницирају са сервером и сама инстанца Линда библиотеке се налази на њему. Даље се може произвољним редоследом повезати произвољан број радних станица са различитих рачунара, и такође се може и повезати Клијентска апликација за задавање задатака. У клијентској апликацији се прво морамо повезати на сервер да би смо могли да покренемо посао. Посао покрећемо тако што дајемо путању до архива до које желимо да извршимо у дистрибуираном окружењу, као и путању до библиотеке за удаљену синхронизацију. Даље задајемо функцију унутар наше архиве од које желимо да почнемо извршавање, обично је то “мејн” функција али не мора нужно да буде. Након тога наводимо класу у којој се налази функција коју желимо да извршимо (Класу задајемо пуним именом пакета). При повезивању радне станице сервер ће слати пинг ради провере активности радних станца и тај пинг ће бити исписан на графичкој конзоли радне станице. Након покретања посла у случају успешног извршавања посла сервер ће вратити резултат посла на клијентску апликацију и тај резултат (испис дистрибуираног програма) ће бити исписан у графичкој конзоли клијентске апликације или Контролног Панела. Код извршавања овог окружења могуће је да се деси да неки контролни панел или радна станица изгубе конекцију према серверу. То није страшно, када се поново успостави конекција резултат ће се исписати на контролни панел у случају да је он изгубио конекцију, ако се десило да радна станица изгуби конекцију, онда корисник мора да прекине посао који се извршава јер је конзистентност података угрожена искључивањем процеса са радне станице која је изгубила конекцију ка серверу. У било ком тренутку можемо да прекинемо посао који се извршава на серверу, и да рестартујемо све радне станице које су повезане на сервер. На серверу можемо да на серверској конзоли испишемо историју послова који су се извршили на том серверу од кад је он кренуо са радом.

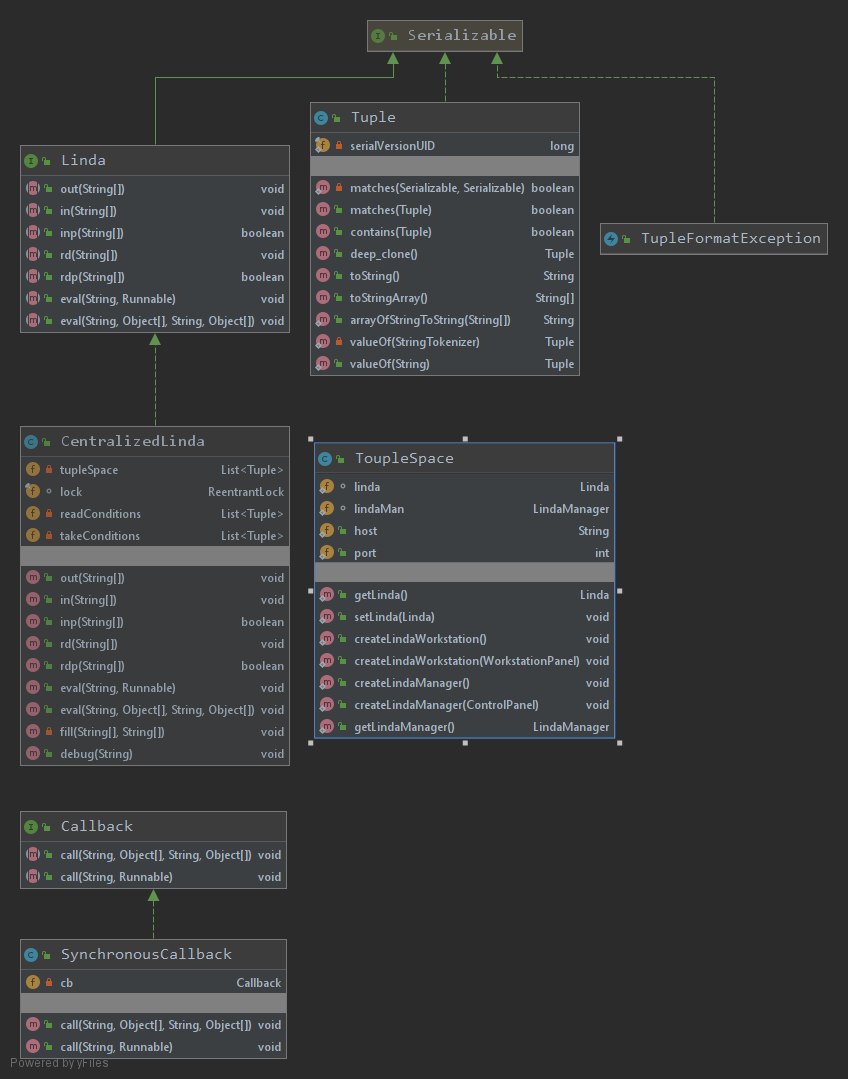
1. **Функционална спецификација система**

Прво ћемо се фокусирати на архитектуру сервера, ако погледамо приложени UML дијаграм можемо да приметимо да имамо 3 класе које су аналогне горе описаним апликацијама. Ове 3 класа пружају потпуну имплементацију серверске и клијентске логике. У датој имплементацији клијент и сервер комуницирају на следећи начин. Клијенти имају инстанцу сервера и шаљу захтеве путем LindaRMI интерфејса, док сервер при регистрацији сваког клијента тражи да му се пошаље и ClientCallbackInterface који чува у хеш мапи са идентификатором клијента и помоћу тог интерфејса контактира клијенте по потреби.

Слика 2. *UML дијаграм Серверског дела програма*



Даље нам је од интереса имплементација саме библиотеке за удаљену синхронизацију Линда. Овде можемо видети на који начин интерагују наша класа CentralizedLina и наша помоћна класа Tuple. Главна идеја око синхронизације саме библиотеке је описана у горњем делу текста, овде ћемо се само осврнути на процес чувања торки, наиме при позиву било које методе Linda интерфејса од прослеђеног низа String-ova се одмах прави објекат торке (прави се инстанца класе Tuple) где се над том инстанцом позива статичка метода arrayOfStringsToString() која од прслеђеног низа стрингова прави један стринг, а потом метода valueOf() врши токенизацију и прави коректне објекте у односу на правила језика Линда.

Слика 3. *UML dijagram Линда дела програма*

1. **Недостаци приложеног решења**

Први и очигледан недостатак је у томе што је ово дистрибуирано окружење централизовано, имамо централизован сервер и централизовану библиотеку за синхронизацију на њему са којом комуницирају све друге компоненте у овом окружењу, тако да у нашем систему имамо једну тачку пуцања а то је централни север. Овај проблем би могли да решимо применом “корд” алгоритма где више не би имали клијент сервер архитектуру, али би наш систем био гео независан и у случају пада неког “нода” у мрежи целокупна функционалност не би била угрожена.

1. **Закључак**

Овај пројекат је јако користан за схватање концепата конкурентног и дистрибуираног програмирања, као и схватања Јава технологија а и принципа објектно орјентисаног програмирања и пројектовања софтвера. Највеће потешкоће у изради оваквих пројеката могу да се јаве у визуелизацији извршавања више нити и проналажења специфичних ситуација у којима програм може да се понаша непредвидиво или чак некоректно.