****

**FACULTATEA DE AUTOMATICA SI CALCULATOARE**

**SISTEME DISTRIBUITE**

**Assignment 3.2**

Asynchronous Communication using Messaging

**Profesor Indrumator: Student:**

**Cioara Tudor Bucea Valentina Sorina**

**Grupa 30244**

CUPRINS

[1. Arhitectura conceptuala a sistemului distribuit 3](#_Toc466818846)

[2. Diagrama UML de implementare 5](#_Toc466818848)

[3. Readme file 6](#_Toc466818849)

[Bibliografie 8](#_Toc466818850)

# Arhitectura conceptuala a sistemului distribuit

Aceasta aplicatie se structureaza pe o arhitectura de tip client-server, in care clientul (browser-ul in cazul de fata) solicita anumite informatii (efectueaza un request), urmand ca serverul sa ii raspunda cererii (printr-un response). Clientul si serverul comunica prin retea, clientul nu impartaseste niciuna dintre resursele sale, doar solicita anumite functii de continut din partea serverului, in functie de necesitatile sale. Prin urmare, clientii initiaza sesiuni de comunicare cu serverele care asteapta sa primeasca request-uri.

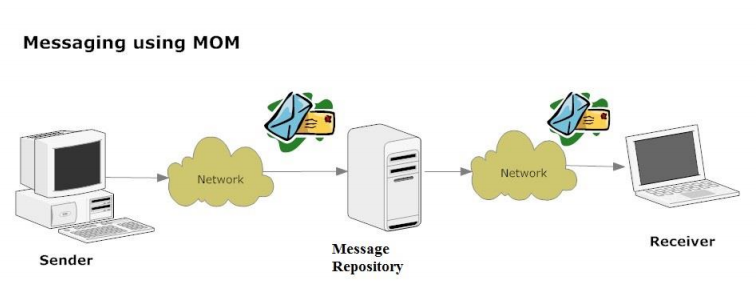
Serverele sunt clasificate in functie de serviciile pe care le furnizeaza, cel pe care aplicatia mea se bazeaza serveste pagini web cu un anumit continut, asta deoarece am construi o aplicatie de tip web ce foloseste servicii de request-reply. Un serviciu este o abstractie a resurselor informatice si un client nu trebuie sa fie preocupat de modul in care serverul efectueaza indeplinirea cererii si livrarea raspunsului. Clientul trebuie doar sa inteleaga raspunsul bazat pe bine-cunoscut-ul protocol de aplicare/comunicare, adica continutul si formatarea datelor pentru serviciul solicitat. Serverul si clientii schimba mesaje intr-un model de mesagerie cerere-raspuns: clientul trimite o cerere, iar serverul returneaza un raspuns. Acest schimb de mesaje este un exemplu de comunicare inter-proces. Pentru a comunica, computerele trebuie sa aiba un limbaj comun, iar acestea trebuie sa respecte regulile, astfel incat atat clientul, cat si serverul stiu ce sa astepte. Limbajul si regulile de comunicare au fost definite pe baza a ceea ce browser-ul stie sa interpreteze si anume cod HTML, in consecinta, server-ul asteapta anumite cereri sub acest format, urmand sa trimita inapoi un raspuns codificat tot in aceasta maniera.

Comunicarea intre client si server se realizeaza prin *Uniform Resource Locator*, aceasta adresa ia forma unui sir de caractere care descrie modul in care se gaseste o resursa pe Internet. URL-urile au doua componente principale: protocolul necesar pentru a accesa resursele (pentru serviciul extern *htp – Hypertext Transfer Protocol*) si locatia resursei (host name-ul, in cazul acestei aplicatii: *localhost*, filename-ul: numele aplicatiei si a servelt-ului Java ce serveste respectivul request, numarul portului: *8080*, datorita server-ului Apache Tomcat ce l-am utilizat). Majoritatea URL-urilor utilizate au fost relative, referindu-ma doar la servlet-ul Java ce va efectua o anumita cerinta, luand ca baza host-ul, port-ul, numele aplicatiei si fisierul de referinta.

*Hypertext Transfer Protocol* este un protocol de aplicatie pentru sistemele informatice distribuite, colaborative, hipermedia, el fiind fundamentul comunicarii de date pentru *World Wide Web*. Este sub forma unui text structurat care utilizeaza legaturi logice (*hyperlink-uri*)intre nodurile ce contin text si de asemenea constituie un protocol de schimburi si transfer de hypertext. *HTTP* este proiectat pentru a permite elementelor retelei intermediare sa imbunatateasca sau sa permita comunicarea intre clienti si servere. Site-urile cu un trafic foarte ridicat beneficiaza adesea de servere cache web care livreaza continut in numele serverelor pentru a imbunatati timpul de raspuns. Datorita faptului ca protocolul *HTTP* si serverele web nu au stare (pentru serverul de web fiecare cerere este o noua solicitare de a procesa si nu poate identifica daca provine de la clientul ce a trimis alte cereri anterioare).

Arhitectura sistemului distribuit se bazeaza pe trei nivele, logica de proces functional, accesul la date, stocarea datelor informatice si interfata cu utilizatorul sunt dezvoltate si mentinute ca module independente si separate. Acest tip arhitectural este un model de design software foarte bine stabilit, permitad oricarui nivel sa fie modernizat sau inlocuit in mod independent. Primul nivel este cel de *Presentation*, aici afisandu-se informatii de top referitoare la serviciile disponibile pe un site web. Aceasta faza comunica cu nivelul inferior prin trimiterea cererilor catre analiza logica, dar si cu browser-ul, prin expedierea rezultatelor. Cel de-al doilea nivel este cel de aplicare, *Business Layer*, numit si nivelul de mijloc, de logica, fiind rezultat din categoria de prezentare. El controleaza functionalitatea aplicatiei prin efectuarea procesarii detaliate a cerintelor de care are nevoie nivelul superior. Ultimul nivel este cel al datelor, *Data Access*, aici aflandu-se modelul structurilor pentru stocarea informatiilor in format txt. Datele din acest nivel sunt mentinute independent de server-ul de aplicatie sau de logica de afaceri.

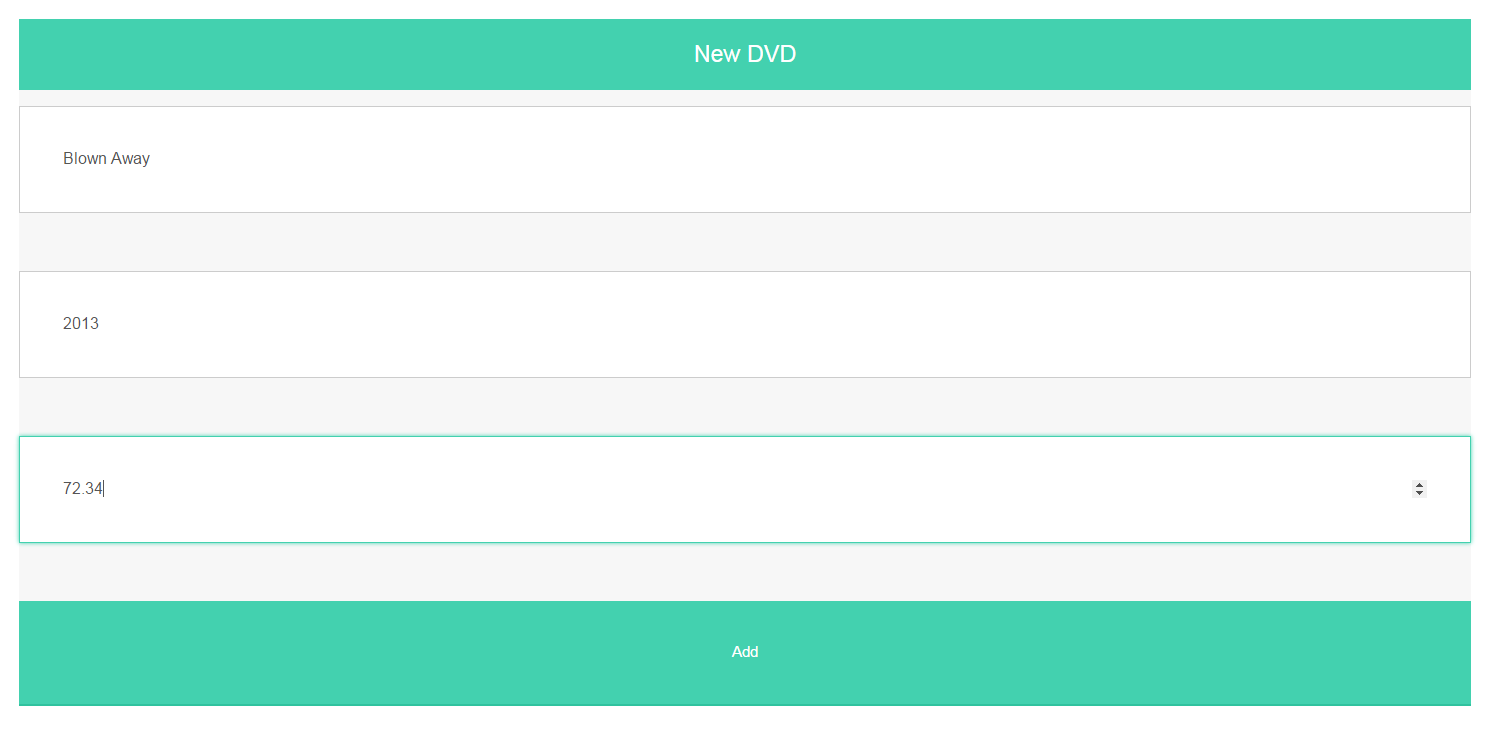
Tipul aplicatiei dezvoltate este de tip *Publish-Subscribe*, avand doua aplicatii: *Producer* si *Consumer*. Acestea folosesc un intermediar si anume un *Exchanger* ce creeaza atatea cozi (in care se pastreaza mesajele dintre producator si consumator), cati consumatori sunt. Datorita acestui tip de comunicare, cu ajutorul unui intermediar, *Producer-ul* si *Consumer-ul* nu sunt dependenti unul de altul, comunicarea dintre acestia realizandu-se asincron.

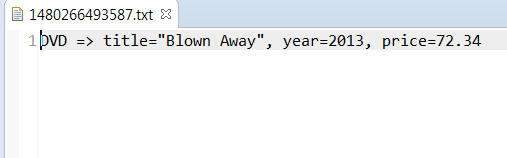


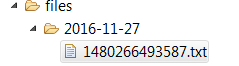
Producatorul introduce, din pagina web, un nou DVD, iar in acest timp, datorita existentei consumatorului ce creeaza un fisier cu detaliile DVD-ului introdus si al celui ce trimite email-uri tuturor clientilor, cu detalii referitoare la aceasta noua achizitie, *Exachanger-ul* creeaza doua cozi ce vor contine fiecare cate un obiect de tip DVD, referitor la cel introdus de administrator. Consumatorii vor citi acest obiect realizand intrebuintarile pentru care au fost creati (in tot acest timp, producatorul executandu-si atributiile sale in mod independent de orice consumator, la fel si fiecare consumator).

# Diagrama UML de implementare

# Readme file

* Seteaza GIT-ul si descarca proiectul de pe <https://SorinaVB@bitbucket.org/SorinaVB/ds2016.git>
  + Creaza un folder local gol in spatiul de lucru de pe calculatorul tau
  + Click dreapta in folder si selecteaza Git Bash
  + Introdu comenzile:
* git init
* git remote add origin <https://SorinaVB@bitbucket.org/SorinaVB/ds2016.git>
* git pull origin master
* Importa proiectul in Eclipse
  + FILE-> Import->Maven-> Existing Maven Projects-> Navigheaza pana la folder-ul cu proiectul descarcat
* Ruleaza proiectul
* click dreapta Assignment3\_2\_PS\_consumerr/start/ConsumerFileStart: Run As -> Java Application
* click dreapta Assignment3\_2\_PS\_consumerr/start/ConsumerMailStart: Run As -> Java Application
* click dreapta pe proiectul Assignment3\_2\_PS\_producerr: Run As -> Run on Server
* Introdu specificatiile DVD-ului
* Apasa butonul *Add* pentru a salva dvd-ul in format txt si a trimite email tuturor clientilor





# Bibliografie

* <https://www.rabbitmq.com/man/rabbitmqctl.1.man.html>
* <https://www.rabbitmq.com/getstarted.html>
* <https://dzone.com/articles/getting-started-rabbitmq-java>