UNIVERSITATEA TEHNICĂ CLUJ-NAPOCA
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
SECŢIA CALCULATOARE

VIZAT DECAN
Prof.Dr.Ing. Sergiu NEDEVSCHI

VIZAT ŞEF CATEDRĂ
Prof.Dr.Ing. Kalman PUSZTAI

Sistem de Supraveghere Video in LAN şi WAN (S.V.L.W.)

Absolvent: **Baciu Ciprian**

1. CONŢINUTUL PROIECTULUI:
   Capitole: Introducere, Fundamentare teoretică pentru domeniul abordat, Specificaţiile şi arhitectura sistemului, Proiectare de detaliu, Testare, Utilizare sistem, Concluzii, Bibliografie, Anexe
2. LOCUL DOCUMENTAŢIEI: **Universitatea Tehnică Cluj-Napoca**
3. CONSULTANŢI: **Ş. l. Ing. Cosmina Ivan**
4. DATA EMITERII TEMEI: **decembrie 2003**
5. TERMEN DE PREDARE: **21 iunie 2004**

CONDUCĂTOR DE PROIECT
Ş. l. Ing. Cosmina Ivan

SEMNĂTURĂ ABSOLVENT
Baciu Ciprian

Cuprins

I. Introducere

1. Motivaţie

În zilele noastre a devenit tot mai acută nevoia de a supraveghea diferite obiective. Însă această supraveghere este necesar să se facă de la distanţă şi nu printr-o pază umană la locul obiectivului. Aşadar monitorizarea vizuală a activităţii dintr-o anumită zonă se putea face foarte uşor prin folosirea unor dispozitive de captură de imagini, adică folosirea de camere video sau camere web conectate la un calculator.

Pentru asigurarea unui astfel de serviciu este clară necesitatea unei aplicaţii multimedia distribuite care să aibă implementate metode eficiente de transfer video, şi eventual audio, "real-time", de la punctele ce monitorizează la cele client ce ascultă şi judecă. S-au încercat de-alungul timpului o serie de soluţii în acest sens, unele chiar foarte reuşite. S-a ajuns chiar la implementarea harware a unor soluţii specializate pe acest tip de activitate. Desigur şi performanţele sunt pe măsură, dar în acelaşi timp şi costul, o asemenea soluţie venind de obicei cu un contract de supraveghere şi pază a obiectivului unde urmează sa fie instalat sistemul respectiv.

Într-o oarecare măsură acest gen de aplicaţii sunt similare cu acelea de video-conferinţă, diferenţe apărând doar în scopul utilizării. Resursele hardware folosite sunt practic aceleaşi, centrul fiind reprezentat tot de calculator şi dispozitivele de capturare a imaginilor. O diferenţa importantă fiind totuşi aceea că la aplicaţii de supraveghere este aproape inutilă folosirea sunetului, faţă de situaţia aplicaţiilor de video-conferinţă în care este absolut necesar ca utilizatorii implicaţi să poată auzi ceea ce li se transmite.

Evoluţia continuă în domeniul realizării de dispozitive ce pot fi conectate la calculator, şi implicit a dispozitivelor ce pot captura imagini şi au facilitatea de "plug and play", şi a îmbunătăţirii posibilităţilor de transfer multimedia au dus la perfectarea posibilităţii de comunicare între utilizatorii de calculatoare, la înmulţirea posibilităţilor de divertisment, dar totodată la o îmbunătăţire simţitoare a capacităţilor de a monitoriza şi supraveghea activităţi din diferite planuri.

Apariţia, în această direcţie, a unui framework cum este JMF (Java Media Framework) [9] de la Sun Microsystems şi perfectarea unui protocol de transfer şi comunicaţie multimedia, RTP (Real-time Transfer Protocol) [14], a adus un plus de simplitate în implementarea aplicaţiilor bazate pe capturarea şi transferul datelor în acest format generic.

Desigur această dezvoltare a fost precedată de o creştere simţitoare a puterii de procesare în cazul calculatoarelor de ultimă generaţie şi bineînţeles mărirea lăţimii de bandă în cazul reţelelor de calculatoare.

Astfel, luând în considerare aceste aspecte şi folosind tot ce era actual în acest cadru, am început documentarea teoretică în scopul realizării unei aplicaţii care să folosească toate aceste beneficii apărute pe piaţa dezvoltării de software. Necesitatea unei astfel de realizări este dată şi de nevoia continuă de reducere a costurilor şi în acelaşi timp poate extinde funcţionalitatea calculatorului. Desigur simplitatea şi lucrul în timp real aduce după sine unele neplăceri, de exemplu calitatea datelor multimedia transferate este scăzută, fiind mai importantă fluenţa comunicării şi o cât mai mică utilizare a reţelei de comunicaţie.

2. Specificarea temei

Lucrarea de faţă se va înscrie într-un context general, al aplicaţiilor de supraveghere ce implică transfer video, dar în acelaşi timp face parte din categoria soluţiilor ieftine, soluţii ce implică doar un minim de necesar hardware, nimic de genul hardware-ului specializat. Necesităţile clar conturate sunt:

- câte un PC gazdă a unei camere web, pentru fiecare locaţie ce urmează a fi monitorizată, supravegheată
- un PC gazdă a serverului ce monitorizează legaturile dintre punctele de supraveghere şi punctele de control
- PC-uri pentru punctele de control

Desigur toate aceste elemente nu sunt dedicate acestei activităţi ele vor putea fi folosite şi în alte activităţi diferite de cea necesară aplicaţiei de supraveghere. Aceste elemente dau şi un grad scăzut costului implicat de folosirea unei astfel de aplicaţii. Vor apărea pierderi de calitate, datorită folosirii partajate a resurselor şi limitării pe care o au în acest domeniu aceste resurse.

În plus la cele enumerate mai sus este necesara folosirea unui server de baze de date, care în mod normal există şi el dinaintea instalării unei aplicaţii de supraveghere. Administratorul va fi solicitat pentru a realiza o structură relaţională de tabele care să dea funcţionalitate deplină aplicaţiei, aceasta interogând baza de date spre a extrage diferite informaţii despre utilizatori şi alte obiecte implicate în sistem.

3. Obiective

Orice aplicaţie software are în spate un proces software de realizare. În cadrul procesului software sunt definite anumite obiective pe care, la sfârşit, în faza de finalizare, aplicaţia trebuie să le atingă. Se conturează doua feluri de astfel de obiective: obiective funcţionale, date strict de funcţionalitatea aplicaţiei, şi obiective nefuncţionale, date de o serie de factori ce nu au consecinţe evidente în bunul mers imediat al lucrurilor.

3.1. Obiective funcţionale

Realizarea unui modul de captură

Acest modul va oferi primul pas în funcţionalitatea globală a aplicaţiei, în cadrul acestui modul realizându-se captura efectivă a imaginii de la dispozitivul de captură, fie o cameră video fie un web-cam. Aici se va realiza practic obiectul multimedia ce urmează a fi înregistrat sau trimis spre un client ce ascultă.

Realizarea unui modul de intercomunicare

Modulul este reprezentat de un media server, care are rolul de a face legătura şi a înştiinţa clientul, ce supraveghează, de existenţa unor module de monitorizare, de captură. De asemenea are rolul stocării, pe disc, a fişierelor multimedia înregistrate de clienţi şi înregistrarea acestora într-o bază de date pentru o mai eficientă funcţionare a aplicaţiei. Va avea capacitatea de a transmite către clienţii, ce supraveghează, orice fişier ce a fost în prealabil înregistrat, pentru a fi reanalizat.

Realizarea clientului ce monitorizează

Reprezintă capătul lanţului funcţional, aici aflându-se judecătorul, cel care analizează ceea ce este surprins de către modulele de captură. La acest nivel se pot lua şi decizii privind înregistrarea a ceea ce este capturat, sau a reanalizării unor capturi anterioare salvate pe discul serverului. De asemenea pot fi surprinse momente din activitatea capturată sub forma unor imagini instantanee. Tot folosind imagini instantanee, snapshot, se poate face o analiză a mişcării existente într-o anumită zonă de captură, ceea ce poate fi de mare folos în cazul necesităţii monitorizării în afara unei activităţi umane în zona de acţiune a obiectului de captură.(Ex.: monitorizare în incinta unei firme după orele normale de program)

Toate acestea formează un ansamblu ce cuprinde, în linii mari, cerinţele funcţionale ale unei aplicaţii din această categorie.

3.2 Obiective nefuncţionale

Obiectivele nefuncţionale reprezintă obiectivele ce nu vor avea un impact imediat asupra produsului, aflându-se în umbra funcţionalităţii afişate. Importanţa lor se resimte, de obicei în timp, după o perioadă de utilizare a produsului ce îl caracterizează.

Corectitudine

Este dată de măsura în care aplicaţia satisface specificaţiile de definiţie şi răspunde obiectivelor formulate de utilizatori. Specificaţiile şi obiectivele unui instrument de supraveghere sunt destul de clare, chiar inutil de a fi specificate de utilizator, deci este imposibilă evitarea satisfacerii acestui factor.

Utilitate

Este absolut evidentă utilitatea unei asemenea aplicaţii, datorită în primul rând necesităţii tot mai acute de evidenţa şi în al doilea rând necesităţii, de asemenea importante, de supraveghere şi monitorizare.

Aplicabilitate

Aplicaţia de faţă prezintă un grad ridicat de aplicabilitate datorita domeniului pe care îl atinge. Orice sistem bine pus la punct va avea suport pentru aplicaţii multimedia şi va putea gestiona aplicaţii de acest gen. Resursele necesare sunt acelea banale şi pot fi puse la dispoziţie aproape în orice condiţii de către cineva ce doreşte utilizarea aplicaţiei.

Portabilitate şi Heterogenitate

Utilizarea în implementare a limbajului de programare Java şi totodată utilizare CORBA (Common Object Request Broker Architecture) dau aplicaţiei portabilitate absolută. De asemenea orice îmbunătăţire ulterioară prin adăugarea de noi module se va putea face cu eforturi minime şi chiar prin folosirea altor limbaje de programare decât Java. Sistemul va putea comunica uşor cu orice alt sistem realizat conform specificaţiilor CORBA.

Fiabilitate

Fiabilitatea este data de posibilitatea de a a-şi executa funcţiile pentru care a fost proiectat un program, cu precizia cerută şi pe o bază consistentă. Posibilitatea de a funcţiona fără erori este de asemenea strâns legată de posibilitatea refacerii din eroare.

Flexibilitate

Dacă la un moment dat se vor schimba cerinţele într-un anume fel, efortul necesar modificării trebuie sa fie cât mai mic, trebuie prevăzute situaţii de completare a cerinţelor iniţiale. Uneori este necesară adăugarea de funcţionalităţi după ce aplicaţiei i s-a dat o formă finală.

Costuri

Costurile de realizare ar putea fi considerate mari datorită folosirii CORBA şi a diferitelor framework-uri, dar această teoretică pierdere este compensată de o creştere deosebită în planul practic şi funcţional.

Costurile de utilizare sunt practic inexistente datorită cerinţelor hardware scăzute ale aplicaţiei. Necesarul este dat de câteva calculatoare şi câteva camere video sau web-cam-uri, ceea ce nu reprezintă, în condiţiile actuale un efort prea mare.

Întreţinere

Uşurinţa cu care se va face o eventuală localizare şi "depanare" a erorilor apărute. Devine necesară prevederea unor mecanisme de logare a activităţii în timp a aplicaţiei.

Reutilizare

Măsura în care un program, sau parte dintr-un program, poate fi reutilizată în alte aplicaţii asemănătoare şi gradul de generalitate al implementării sale dau aplicaţiei un grad de reutilizare. Tot ceea ce nu are directă legătură cu interfaţa utilizator este reutilizabil, fiind realizat în conformitate cu şabloane arhitecturale şi de implementare.

Integritate

Măsura în care se poate controla accesul unui personal neautorizat, atât la date cât şi la program. Aplicaţia prevede un mecanism de autentificare, ce restrânge posibilitatea accesului neautorizat.

Interoperabilitate

Este dată de efortul necesar interconectării cu un alt sistem. In acest sens sistemul propus are un grad foarte ridicat de interoperabilitate, datorită folosirii CORBA.

Eficienţa

Raportul munca – rezultate dă întotdeauna, în orice domeniu, o notă reprezentativă pentru eficienţa produsului realizat. Munca puţina cu rezultate deosebite reprezintă un ideal imposibil, în general cantitatea în muncă este asociată cu calitatea în rezultate, fiind necesară o bună organizare a procesul de realizare.

Uşurinţa în utilizare

Uşurinţa cu care operatorul va reuşi să îşi însuşească cunoştinţele necesare folosirii produsului. Aplicaţiile ce oferă o interfaţă utilizator trebuie să o realizeze cât mai explicit pentru ai oferi utilizatorului posibilitatea să se acomodeze cât mai uşor.

Calitatea produsului

Unul dintre cele mai importante obiective ce nu ţin de funcţionalitatea aplicaţiei, dar la un moment dat ar putea să o afecteze, este acela al calităţii produsului software realizat. Se poate discuta acest factor din două puncte de vedere: unul este acela al stabilităţii produsului software şi al doilea acela al calităţii codului implementat.

Stabilitatea aplicaţiilor software apare a fi foarte importanta de aceea este necesar sa se obţină un produs stabil, cu atât mai mult cu cât o aplicaţie de supraveghere rulează practic non-stop. Pentru a ajunge la această performanţă trebuie făcută o mare muncă de testare asupra produsului considerat final şi reparate eventualele greşeli.

Importanţa calităţii codului implementat se simte în momentul reluării aplicaţiei pentru a fi dezvoltată. Cu cât este codul scris mai “elegant” cu atât îi va fi mai uşor celui ce îl citeşte să îl înţeleagă şi să îl reutilizeze dacă este nevoie. Comentariile au rolul lor într-un cod bine scris.

O altă abordare ar fi aceea dată de “American Society for Quality Control” care sustine că noţiunea de calitate în software presupune doua elemente: conformare la specificaţia produsului, gradul de uşurinţă în utilizare.

Practic noţiunea de calitate a produsului software înglobează toate obiectivele enumerate până acum.

II. Fundamentare teoretică pentru domeniul abordat

Prealabil dezvoltării propriu-zise a unui produs, fie el şi software, trebuie cunoscute în mare măsură uneltele ce urmează a fi folosite în dezvoltare. Astfel este necesară o muncă de cercetare teoretică a tot ceea ce va fi utilizat mai târziu. Cel ce realizează proiectul trebuie să cunoască factorii de calitate ai unui produs, ca şi cel ce încearcă să îl realizeze. Trebuie să îşi fixeze un plan de acţiune în aşa fel încât eventualele modificări sa nu de-a peste cap întreg procesul. Conducătorul proiectului va fixa un anume referenţial bibliografic pe care în timp va încerca să îl acopere.

Este necesar ca baza de cunoştinţe să fie consistentă, dacă se doreşte obţinerea unui produs de calitate care să aibă succes pe piaţă. Aceste cunoştinţe se dobândesc în timp în urma a ore de studiu şi urmând un program bine pus la punct. Ideea realizării unui produs apare din cunoaşterea nevoilor pieţei, în domeniu, coroborată cu o cunoaştere a posibilităţilor produselor asemănătoare celui ce urmează a fi realizat.

La baza oricărui produs software stă un proces software, adică un set de activităţi care permit transformarea cerinţelor utilizatorului în produsul final. La baza acestui proces stă o susţinută fundamentare teoretică care să cuprindă tot ceea ce urmează să fie folosit, dar uneori şi ceea ce nu se foloseşte efectiv ajutând doar la extragerea unor concluzii importante. Documentarea trebuie să aibă un caracter comparativ, se va folosi produsul ce satisface cel mai bine nevoile, dar pentru aceasta trebuie cunoscute mai multe produse similare.

Chiar dacă la final partea de documentare nu se vede şi nu se simte, este foarte importantă, reprezentând punctul de plecare. Realizarea unei unelte software care să poată asigura funcţionalitatea unui sistem de supraveghere necesită urmărirea câtorva paşi în dezvoltare, dar mai întâi o documentare prealabilă consistentă.

1. Sisteme distribuite

.1. Generalităţi

O definiţie simplă apare astfel: o colecţie de resurse heterogene interconectate într-o reţea şi care suportă dezvoltarea de aplicaţii şi servicii care folosesc arhitectura fizică a acestor resurse. Trebuie avut în vedere faptul ca fiecare componentă a sistemului este diferită

într-un fel sau altul, dar trebuie sa coopereze prin diferite metode: schimb de mesaje, migrare de obiecte, întârzieri etc. Componentele sunt autonome, nu îşi partajează memoria fiecare având acces la memoria sa locală. Un sistem distribuit poate fi de mai multe feluri, bazat pe diferite modele: Modelul Client-Server, Modelul obiectual, Modelul "code on demand".

Definiţie. *Un model este o reprezentare simplificată a sistemului în care sunt surprinse parte din caracteristicile cunoscute , inferate sau solicitate cu scopul abstractizării lor în intenţia de a studia sau defini alte caracteristici*.

Având în vedere necesitatea unui instrument de supraveghere de a monitoriza activitatea de la distanţă, este evidentă necesitatea mai multor module în cadrul unui astfel de sistem, deci a distribuirii acestuia. Modelul client-server şi cel obiectual sunt folosite în aplicaţia de faţă.

.1.1. Modelul Client/Server

În cazul modelului client/server activitatea are loc în mod asimetric: un proces cere informaţia (clientul), iar altul o are şi o trimite (serverul), după o prealabilă validare. În cazul în care serverul are nevoie de informaţii de la un alt server, devine el însuşi client.

În cazul sistemului de supraveghere un client poate să ceară serverului să ii trimită un anume fişier înregistrat pe disc la un moment dat, sau un alt tip de client poate cere permisiunea de a începe o sesiune de transfer de fişiere. În funcţie de validarea clientului acesta va primi sau nu răspunsul dorit. Un server trebuie sa fie capabil să răspundă la mai multe cereri din partea clienţilor, procesul de cerere-răspuns să nu fie blocant, o eventuală cerere de la un client să nu fie blocată până când se va răspunde cererii precedente. Se va folosi multithreading, pentru fiecare client se deschide un nou fir de execuţie pentru a-l deservi, astfel serverul rămâne deschis spre a primi noi cereri.

Aşadar procesul este foarte simplu: clientul trimite o cerere, serverul ia cererea, ii verifică valabilitatea, procesează informaţiile şi apoi trimite răspunsul.

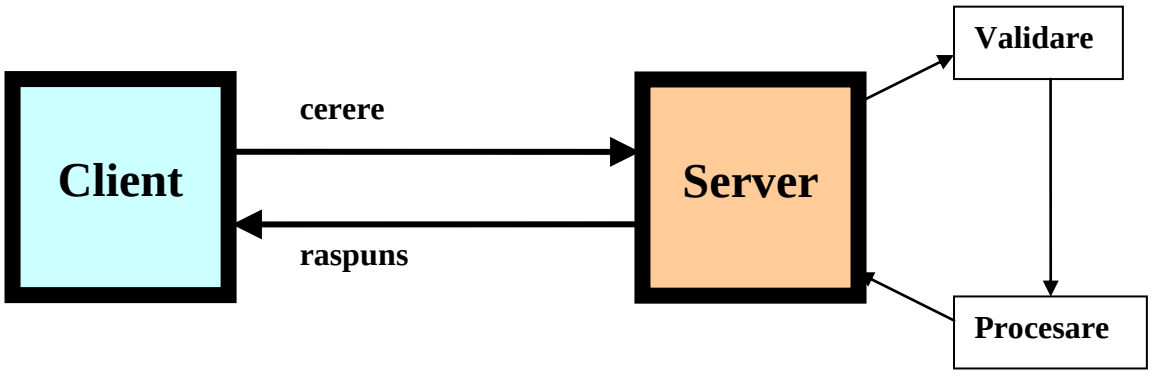

Figura 1 - Modelul Client/Server

.1.2. Modelul Obiectual

Fiecare resursă împărtăşită este văzută în sistem ca un obiect. Un obiect poate fi atât client cât şi server, poate să ceară servicii de la un alt obiect, dar poate să i se ceară şi lui servicii. Într-un sistem distribuit obiectual fiecare obiect are identitate unică, identitate ţinută de un server de nume. Obiectele pot să migreze oriunde în sistem, dar trebuie să-şi păstreze identitatea, de aici şi necesitatea unui server de nume. Migrarea obiectelor implică migrarea claselor, partea de identificare a locaţiei şi a comunicării între obiecte este realizată de către middleware. Există câteva metode de a folosi obiecte la distanţa în sistemele distribuite: RMI (Remote Method Invocation), Web Services, CORBA. O comparaţie a acestor metode se va face în următoarele părţi din lucrare.

.2. Specificaţii CORBA pentru sisteme distribuite

"Common Object Request Broker Architecture" este o arhitectură ce dă posibilitatea programatorilor de a realiza aplicaţii interoperabile bazate pe obiecte distribuite.

OMG, Object Management Group, este o organizaţie de peste 800 de companii fondată în 1989 şi care promovează folosirea tehnologiei orientate obiect în aplicaţiile software. Ţelul organizaţiei este acela de a realiza o specificaţie generală şi un framework comun pentru dezvoltarea de aplicaţii, dar şi reutilizarea, portabilitatea, interoperabilitatea aplicaţiilor software obiectuale în sisteme distribuite şi heterogene.

.2.1. Generalităţi

Cheia înţelegerii CORBA este Modelul de Referinta, care are următoarele componente:

- **Object Request Broker** (ORB) face ca obiectele să trimită şi să primească cereri, răspunsuri, în mod transparent, într-un sistem distribuit. Reprezintă fundamentul construirii de aplicaţii obiectuale distribuite şi interoperabilităţii între aplicaţii pe sisteme hetero- si omogene.(Ex: ORBacus, JavaORB)[10]
- **Object Services**, este o colecţie de servicii, interfeţe şi obiecte, care suportă funcţiile de bază pentru folosirea şi implementarea de obiecte. Serviciile sunt necesare pentru construirea oricărei aplicaţii distribuite şi sunt independente de domeniul aplicaţiei.
- **Common Facilities**, este o colecţie de servicii care poate fi împărţită de mai multe aplicaţii, dar nu are aceeaşi importanţa fundamentală precum “Object Services”
- **Application Objects**, sunt produse ale aceluiaşi producător, care controlează interfeţele lui, nu sunt standardizate de OMG [13]

ORB-ul reprezintă centrul modelului de referinţă asigurând comunicaţia specifică între aplicaţii CORBA.

- **Interface Definition Language**, (IDL), este un limbaj de specificaţii care dă posibilitatea de separare a interfeţelor de implementarea obiectelor. Aduce un set de tipuri de date ce pot fi folosite pentru definirea altora şi mai complexe, ce vor fi folosite pentru definirea capacităţilor sistemului distribuit în cauză. De asemenea specificaţiile IDL nu depind de limbajul de programare folosit în scrierea aplicaţiei.[11]

CORBA reprezintă o metodă de a realiza aplicaţii distribuite şi de a apela obiecte la distanţă, spre a realiza comunicarea între diferite module ale unei aplicaţii. Comunicaţia între modulele unei aplicaţii putea fi realizată şi folosind tehnologia de nivel jos, socket având posibilitatea de a trimite obiecte prin serializare, dar cantitatea de informaţii ce trebuiau trimise, în scopul comunicării era foarte mare, ceea ce ar fi dus la o scădere a performanţei, dată de timpii de răspuns la cereri mari. Astfel este indicată folosirea apelului la distantă pentru a scădea timpii de răspuns. Sunt situaţii pentru care tehnologia socket este mai utilă, cum ar fi acelea în care este necesar transferul în siguranţă deplină a unor date reprezentând diferite fişiere de dimensiuni mari. Tehnologii precum CORBA se bazează pe un model obiectual, în care orice sistem obiect este un server ce oferă serviciilor clienţilor.

Precum RMI, CORBA simplifică munca programatorului datorită abstractizării între aplicaţia sa şi nivelul programării pe reţea. CORBA oferă posibilitatea dezvoltării de aplicaţii orientate obiect, în care obiectele la distanţă sunt instanţiate şi metodele lor folosite ca şi cum ar fi pe acelaşi calculator. Obiectele pot fi implementate în orice limbaj de nivel înalt, precum C++ sau Java, pentru care exista un compilator IDL. Aici apare diferenţa majoră faţa de RMI care deşi oferă aceleaşi facilităţi când este vorba de folosirea ca limbaj de programare şi platforma Java, dar nu mai oferă independenţa folosirii altor limbaje de programare şi platforme în afara celor oferite de Java. Astfel orice aplicaţie ce integrează module scrise în diferite limbaje de programare nu va mai avea compatibilitatea necesară folosirii RMI, ca metodă de acces a obiectelor la distanţă.

CORBA oferă acea independenţă a interfeţei obiectului de implementarea sa, ceea ce ii conferă şi independenţa de limbajul de programare folosit în implementarea propriu-zisă a obiectului. Astfel pot fi construite sisteme ce să cuprindă diferite module, iar în funcţie de ce funcţionalitate trebuie să aibă, se poate alege un limbaj de programare care satisface cel mai bine condiţiile de realizare.

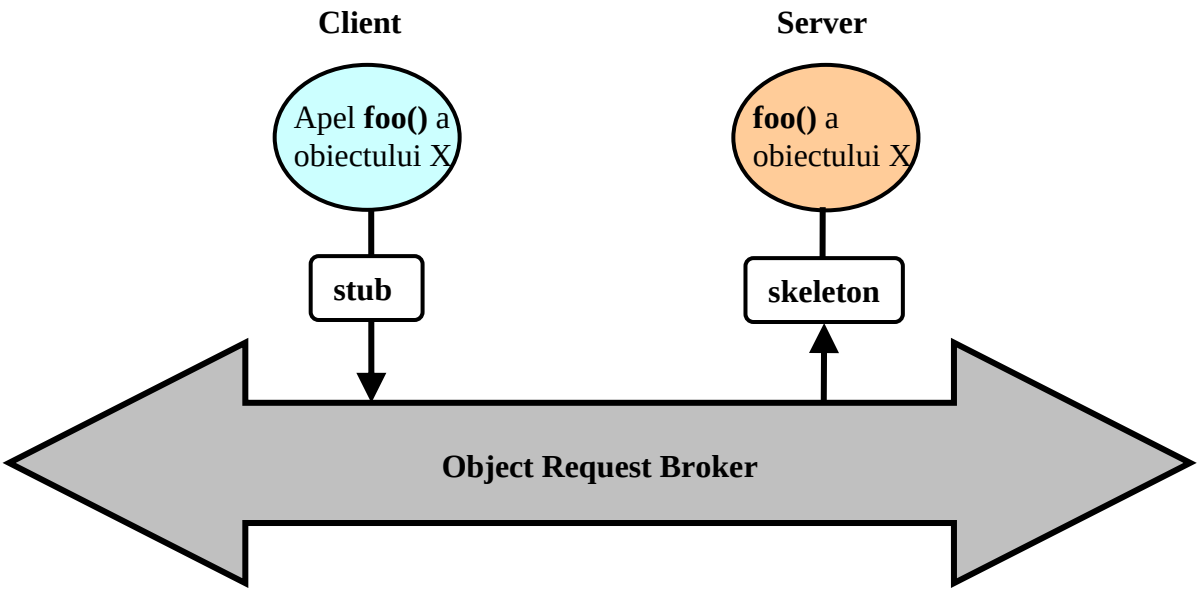

Figura 2 - Client CORBA invocând o metoda a unui obiect la distanţă

.2.2. Object Request Broker – ORBacus

Figura de mai jos descrie un obiect ce trimite o cerere către implementarea obiectului. *Clientul* este entitatea care încearcă să facă o operaţie pe obiect, iar *Implementarea Obiectului* reprezintă codul şi datele care definesc în fapt obiectul.

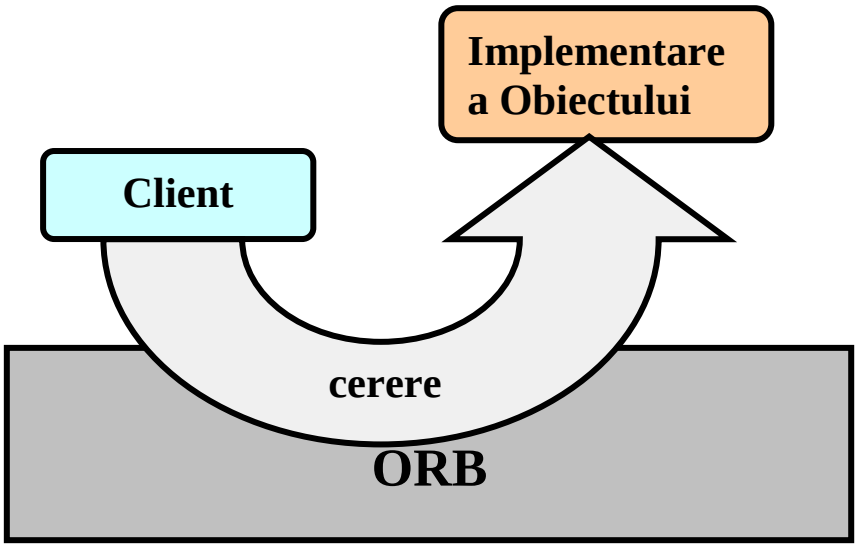

Figura 3 - O cerere trimisă prin ORB

ORB-ul este responsabil pentru toate mecanismele necesare găsirii implementării pentru obiectul către care s-a făcut cerere, pregătirii implementării obiectului pentru a primii cererea şi de a comunica datele necesare rezolvării cererii. Interfaţa văzută de client este absolut independentă de locaţia efectivă a obiectului, de limbajul de programare în care e scris obiectul, sau orice alt aspect ce nu reiese din interfaţa obiectului.

Pentru a face o cerere clientul poate să folosească interfaţa dinamică (independentă de interfaţa obiectului chemat) sau stub-ul OMG IDL (stub-ul specific ce depinde de interfaţa obiectului chemat). De asemenea clientul poate interacţiona direct cu ORB-ul pentru diferite funcţii. Implementarea obiectului poate primi cererea prin interfaţa dinamică sau interfaţa IDL.

Desigur şi ORB-ul poate să difere, astfel ca ORB-uri diferite va da implementări diferite coroborate cu diferite compilatoare de IDL şi adaptoare obiect. Este posibil ca la un moment dat un client să aibă acces la mai multe referinţe controlate de diferite ORB-uri. Când doua ORB-uri sunt conectate spre a conlucra, ORB-urile trebuie să poată să îşi deosebească referinţele ce le aparţin. Nu este responsabilitatea clientului să facă acest lucru.

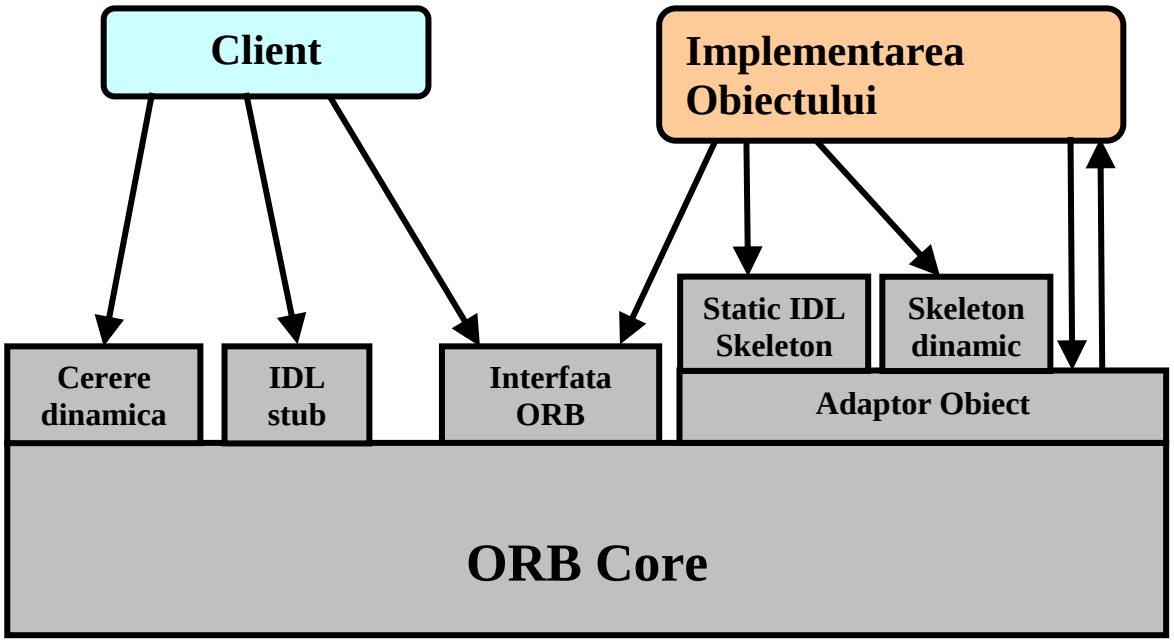

Figura 4 - Structura interfetelor

ORBacus este un ORB care corespunde specificaţiilor CORBA definite în: *"The Common Object Request Broker: Architecture and Specification"*, *"C++ Language Mapping"*, *"IDL/Java Language Mapping"* [11], and *"Portable Interceptors"*.

Are câteva calităţi care îl fac unul dintre cele mai folosite pe piaţa dezvoltatorilor de aplicaţii software distribuite:

- este gratis, nu se solicită nici un fel de licenţă plătită pentru folosirea lui
- conferă o infrastructură pentru aplicaţii de performanţă si scalabilitate ridicată
- standarde pentru a evita blocarea pe produse ale unui anume producator
- este un produs ajuns deja la maturitate

.2.3. Servicii CORBA

Serviciile obiectelor CORBA reprezintă sunt o colecţie care suportă funcţii de bază pentru folosirea şi implementarea obiectelor. Aceste servicii conţin obiecte şi interfeţe ce sunt necesare pentru a construi o aplicaţie distribuită. Ele sunt independente de tehnologia folosită pentru a implementa obiectele şi interfeţele ce le oferă şi de domeniul aplicaţiei în care vor fi folosite. Câteva dintre aceste servicii sunt:

- **Serviciul de nume**

Pentru a putea folosi o cerere de metoda a unui obiect, aplicaţia are nevoie de o referinţă la acel obiect, iar această referinţă poate fi legată de un nume folosind acest serviciu. Serviciul de nume dă posibilitatea de a lega un obiect de un nume folosind un context de nume. În acelaşi sistem CORBA, pot fi implementate mai multe servicii de nume.

- **Serviciul de evenimente**

În programarea orientată obiect evenimentele sunt folosite pentru a aduce la cunoştinţa altor obiecte ca au avut loc diferite acţiuni. În aplicaţiile non-distribuite un ascultător de eveniment este adăugat la un obiect, după care ascultătorul primeşte evenimente.

Serviciul de evenimente defineşte două roluri pentru obiecte: rolul de producător şi rolul de consumator. Producătorii produc date eveniment, iar consumatorii procesează datele eveniment.

Aplicaţia de supraveghere foloseşte pentru a simula evenimentele, tehnica "callback". Deoarece erau puţine situaţii în care era necesară înştiinţarea altor obiecte de anume acţiuni, nu se justifică folosirea acestui serviciu.

- **Serviciul de timp**

Conţine practic doua servicii, *Serviciul de timp* şi *Serviciul eveniment de timp*, şi oferă următoarele facilităţi: utilizatorul poate să obţină timpul curent, pot fi generate evenimente bazate pe timere şi alarme, timpul dintre două evenimente poate fi calculat.

- **Serviciul de schimb**

Serviciul de schimb facilitează oferirea şi descoperirea instanţelor serviciilor de diferite tipuri. Un obiect de acest fel poate fi văzut ca un obiect prin care alte obiecte pot să-şi publice capacităţile şi să îşi potrivească nevoile.

- **Serviciul de proprietate**

Oferă posibilitatea de a defini proprietăţi pe care operaţiile şi atributele obiectelor le au. Proprietăţile sunt valori scrise şi numite care sunt asociate dinamic cu un obiect CORBA.

- **Serviciul timp de viaţa**

Defineşte diverse convenţii pentru crearea, ştergerea, copierea şi mutarea obiectelor.

.2.4. Serviciul de nume

După cum l-am descris mai sus acest serviciu oferă posibilitatea de a lega un nume de fiecare obiect la distantă. Pentru aceasta acel obiect trebuie înregistrat într-un context de nume, reprezentat de un obiect ce conţine mai multe legături de nume, reprezentând perechile nume-referinţa obiect, şi desigur fiecare nume este unic. O aplicaţie poate folosi acest serviciu pentru a găsi referinţă la obiect pentru un anume nume. Serviciul de nume returnează o referinţă a obiectului asociat numelui, care poate fi folosită pentru a invoca metode ale acelui obiect. Pentru a localiza serviciul de nume CORBA, în timpul iniţializării ORB-ului, trebuie cunoscute adresa platformei pe care rulează şi portul pe care ascultă. Aceste valori pot fi transmise obiectelor ce vor să folosească serviciul, printr-un fişier de configurare sau alte resurse statice.

Prin folosirea acestui serviciu, manipularea numelor este simplificată şi obiectele CORBA pot fi localizate fără să li se cunoască referinţa. Referinţa obiectelor poate să se schimbe fără să fie necesare modificări la clienţii serviciului de nume. Este de reţinut faptul că la un obiect pot fi legate mai multe nume, dar acestea trebuie să respecte regula de a fi unice în contextul în care se face legătura.

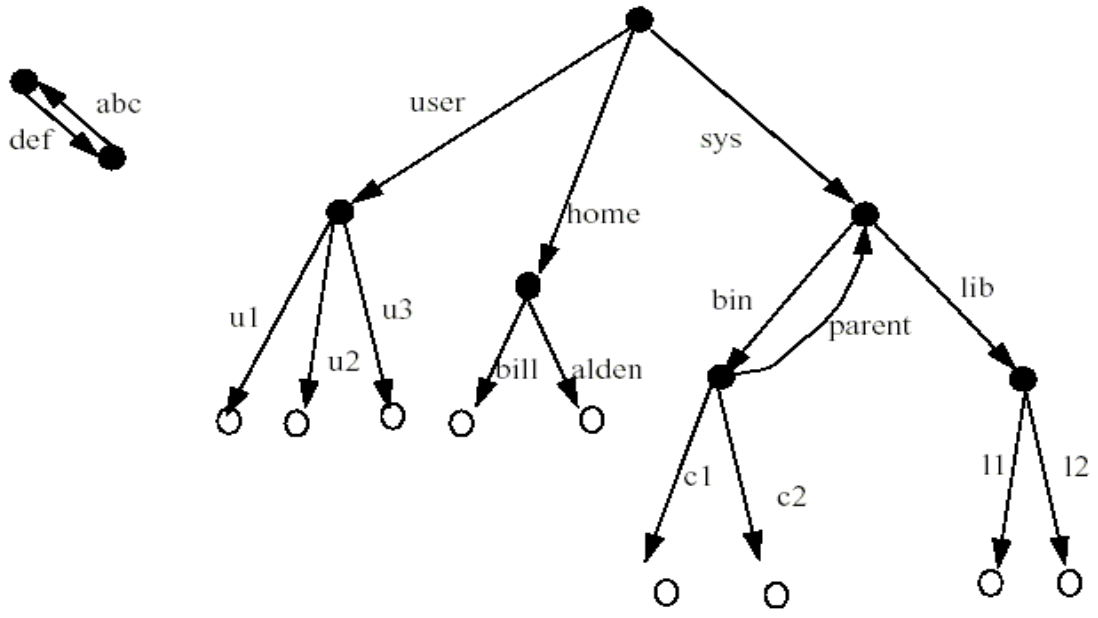

Figura 5 - Graf de nume

.2.5. Mapare IDL-Java

Specificaţii privind maparea CORBA la limbajele de programare există pentru următoarele limbaje: Ada, C, C++, COBOL, Java.

IDL reprezintă limbajul de descriere al interfeţelor obiectelor ce vor putea fi folosite de la distanţa, astfel ca este necesar sa fie independent de tehnologia ce va fi folosită pentru a implementa obiectul propriu-zis. Pentru a efectua compilarea interfeţelor spre a le folosi va trebui folosit compilatorul specific limbajului în care este făcută implementarea obiectelor ce urmează a fi accesate la distanţă.

Maparea la Java este strict dependentă de specificaţiile setului de pachete standard conţinut în org.omg.*. În general identificatorii şi numele din IDL sunt mapate în Java fără a fi schimbate, însă dacă ar putea apărea o coliziune de nume aceasta este rezolvată prin adăugarea unui *underscore* la numele mapat. De exemplu o interfaţă O*bjInterface* va fi mapata la interfeţele Java O*bjInterface* şi O*bjInterfaceOperations*, şi clasele adiţionale O*bjInterfaceHelper,* O*bjInterfaceHolder,* O*bjInterfacePOA,* opţional O*bjInterfacePOATie.* Dacă mai mult de un sufix rezervat este prezent în numele IDL atunci va fi adăugat la nume un underscore.

Tipurile de date folosite de IDL pot fi foarte uşor mapate în Java, fiind cele standard, de asemenea folosirea constantelor. IDL are suport pentru definirea de excepţii şi chiar se pot extinde tipurile de bază prin definirea în fişierul ".idl" a unor tipuri extinse.

Practic programatorul trebuie doar să scrie un fişier de specificaţii pentru interfeţele obiectelor ce se vor folosi la distanţă, respectând sintaxa şi cerinţele IDL, iar apoi totul rămâne în grija compilatorului, să facă maparea, să rezolve probleme de nume în aşa fel încât să producă necesarul de clase Java. După această implementarea propriu-zisă a obiectelor descrise de interfeţe se va face în modulul care le poate da funcţionalitate.

.3. Java şi JMF - suport Java pentru aplicatii multimedia

.3.1. De ce Java?

Toate limbajele de programare reuşesc într-o mai mică sau mai mare măsură să abstractizeze realitatea problemei, să facă asocierea între modelul maşină şi modelul problemei ce urmează a fi rezolvată. S-a pornit de la limbajul de asamblare care dă o

abstractizare destul de mică, apoi s-a continuat cu FORTRAN, BASIC, C care sunt abstractizări ale limbajului de asamblare şi s-a ajuns la limbaje de programare orientate obiect care dau programatorului posibilitatea de a-şi defini elemente în domeniul problemei. Între acestea din urmă, Java, ca orice limbaj uman, dă posibilitatea exprimării de concepte.

- este un limbaj de programare neutru arhitectural, "Write once, run anywhere" (Sun Microsystems motto), facilitate ce ii dă o portabilitate totală
- este dinamic, încărcare dinamică a claselor de către interpretor la momentul execuţiei
- este simplu de folosit, necesită doar cunoaşterea modelului orientat pe obiecte
- este un limbaj robust, introducerea excepţiilor şi a mecanismelor de tratare a lor , eliminarea pointer-ilor, garbage collection automat
- este un limbaj sigur, implementat sistem de nivele de securitate
- suportă multiple fire de execuţie, multithread
- are utilitar de realizare a documentaţiei (javadoc)

([1], [2], [3], [4])

.3.2. Specificaţii multimedia

Odată cu dezvoltarea domeniului telecomunicaţiilor s-a ajuns la necesitatea ca aparate de comunicare de dimensiuni mici si cu capacitate de stocare relativ mică, dar putere de prelucrare mare, să ofere posibilitatea de a captura, transmite, recepţiona şi prelucra imagini fotografiate respectiv imagini capturate de o cameră video. Odată cu dezvoltarea metodelor de compresie video, respectiv dezvoltarea reţelelor de comunicaţie şi răspândirea reţelelor de comunicaţie cu lăţime de bandă mare s-a creat posibilitatea transmiterii informaţiei vizuale în timp real. Interesul pentru tehnologiile de capturare, transmitere/recepţie a informaţiilor video între puncte îndepărtate din punct de vedere fizic, respectiv a prelucrării/redactării informaţiilor video a crescut semnificativ. Domeniul de utilizare a acestor tehnologii s-a dezvoltat, dintre acestea aş aminti sistemele de securitate bazate pe camere de luat vederi şi înregistrări, sistemele de conferinţa video, bazele de date de imagini video (video storehouse), telefonia mobilă din generaţia a 3-a care oferă conţinut multimedia (inclusiv în format video), sisteme utilizate în domeniul medicini, sisteme folosite în domeniul educaţiei sau a administraţiei publice. În unele ţari firmele de cablu TV oferă

servicii de video-on-demand, oferind clienţilor posibilitatea de a viziona filme/programe la comandă, contra unei sume de bani. Sau dezvoltat o serie de echipamente numite SetTopBox care prin intermediul unui televizor pe lângă posibilitatea de a alege dintre canalele de televiziune emise, oferă posibilitatea de a viziona programele/filmele din catalogul firmei TV-cablu în regim "la cerere", respectiv dispune de suport pentru navigare pe internet.

Cu toate aceste îmbunătăţiri din ultimul timp, un transfer de date multimedia în timp real solicită destul de mult reţeaua, ceea ce duce în cazul unui trafic multimedia mare la probleme de supraîncărcare a reţelei. Şi această problemă se poate rezolva prin utilizarea de codificatoare, respectiv decodificatoare, dar trebuie asumată responsabilitatea pierderii în calitate. Folosirea în transferul de date multimedia a unor protocoale specializate, cum este RTP, reduce din problemele binecunoscute ale acestui fel de comunicaţie. În acelaşi timp au fost impuse câteva standarde în tehnologia multimedia.

Audio/Video Streaming

Audio/Video streaming presupune trimiterea unor date în format video prin reţea, intranet sau internetul, de la un server media spre un client. Serverul împarte fişierul video în pachete care la destinaţie sunt reasamblate de către client şi fişierul e derulat în paralel cu sosirea datelor. Video streaming diferă de un simplu transfer de fişiere prin faptul că fişierul de date este derulat în timp ce acesta soseşte prin reţea, fără să fie necesară stocarea lui pe harddisk-ul clientului. Un stream poate fi trimis în mod "one-to-one" (unicast) sau în mod "one-to-many" (multicast). Privind procesul de video streaming din ansamblu putem identifica 5 componente majore in realizarea unui sistem video streaming:

- Utilitare folosite în procesul de codificare a datelor video/audio
- Formatele de date folosite pentru video/audio streaming
- Serverele multimedia
- Retelele prin care datele sunt distribuite
- Programele de vizualizare a datelor receptionate la partea de client

Standarde non-ISO pentru video streaming

- **ITU E.711** – standard audio pentru stream de 64K, utilizează compresie A-Law sau mu- Law PCM. A fost folosit ca H.323 audio şi sisteme de video-conferinţă.

- **ITU E.722** – standard audio pentru stream de 64K, utilizează compresie ADPCM (Adaptive Differential Pulse Code).
- **ITU E.728** – standard audio folosind compresia LD-CELP, stream de 16Kbps.
- **ITU H.225.0** - identic cu ISO/IEC RTP.
- **ITU H.261** – format pentru compresia unei transmisii de video-conferinţa de tip H.320. Stream cu bit-rate multiplu de 64Kbps. A fost formatul de referinţă folosit la dezvoltarea standardului MPEE.
- **ITU H.263** - format pentru compresia unei transmisii de video-conferinţa. Oferă rezoluţii şi frame-rate superior celui oferit de ITU H.261.
- **ITU H.320** – folosit în sisteme de video-telefoane şi echipamente de tip terminal de lăţime de bandă mică. Standard ITU pentru video-conferinţe folosind linii digitale telefonice, ISDN. 1999
- **ITU H.323** – standard pentru voice- şi video-conferencing prin reţele LAN respectiv Internet. Este folosit în telefonia IP, permite transportul oricărui combinaţii de audio, video şi date. Poate utiliza mai multe standarde de format audio/video, incluzând H.261, H.263 (pt video), E.711, E.723.1. Oferă suport pentru gateway
- **ITU T.120** – e un standard care include o serie de specificaţii care acoperă toate aspectele unei conferinţe de date. (T.121 – Generic Application Template, T.122 – Multipoint Communication Service, T.123 – Audiovisual Protocol Stacks, T.124 – Generic Conference Control, T.125 – Multipoint Communication Service, T.126 – Still Image &Annotation Protocol, T.127 – Multipoint Binary Transfer, T.128 – Audio Visual Control, T.130 – Realtime Architecture, T.131 – Network Specific Mappings, T.132 – Realtime Link Management, T.133 – Audio Visual Control Services, T.RES – Reservation Services, T.Share – Application Sharing Protocol, T.TUD – User Reservation)
- **W3C SMIL** – format pentru descrierea unor prezentaţii. Aceste prezentaţii pot conţine streamuri audio, video, imagini, text sau orice alt format multimedia.

Standarde ISO pentru video streaming

Principalele formate video standardizate de către institutul de standardizare ISO/IEC sunt: MPEG1, MPEG2, MPEG4, MPEG7, MPEG21.[18]

- **MPEG1** – reprezintă un format video cu bitrate constant pentru aplicaţii care necesită bitrate mic şi rezoluţii mici.

- **MPEG2** – reprezintă un format video destinat aplicaţiilor cu bitrate mare şi rezoluţie mare, reprezintă standardul de facto în domeniul televiziunii digitale. Stă la baza tehnologiei de filme DVB.
- **MPEG4** – reprezintă versiunea îmbunătăţită a formatului MPEG, cu suport pentru conţinut interactiv, drepturi de autor
- **MPEG7** – numit şi „interfaţa de descriere a conţinutului multimedia" (Multimedia Content Description Interface) oferă un set bogat de unelte de descriere respectiv interogare, organizare a conţinutului multimedia.
- **MPEG21** – este un cadru multimedia (Multimedia Framework) bazat pe 2 concepte esenţiale: conceptul de „*entităţi digitale*", care sunt unităţi elementare de distribuire şi de tranzacţionare a informaţiei de tip multimedia.

Protocoalele standardizate sunt: SDP, RTP/RTCP, RTSP, SIP.

- **Protocolul SDP** (Session Description Protocol) prezintă un format de descriere a sesiunilor de transmitere de informaţii multimedia.
- **Standardul RTP** (Real time Transfer Protocol) descrie modul de împachetare a datelor video şi contribuie la calitatea şi siguranţa transmiterii acestuia. [14]

RTP este protocol de transfer de date, folosit în transferul datelor de timp real, respectiv audio, video media. Suportă transfer spre mai multe destinaţii folosind adrese multicast, sau prin clonarea sursei de date de la capătul transmiţător al conexiunii. De reţinut este faptul ca nu are nici un fel de mecanism de asigurare a transferului oportun sau de asigurare a calităţii serviciului, aceste probleme rămân în sarcina nivelelor de mai jos. Nu asigură sub nici o formă transmiterea pachetului, dar totuşi adaugă pachetului un număr de secvenţă ce dă destinatarului posibilitatea de reconstrucţie, determinând exact locul pachetului în secvenţa de date primită. Protocolul nu este limitat la aplicaţii bazate pe transfer multimedia, este la fel de util şi în cazul salvării datelor continue, aplicaţiilor distribuite interactive sau acelor de control şi măsurare. Apare totodată şi un protocol de control RTCP (Real-time Transfer Control Protocol) pentru a monitoriza calitatea serviciului şi a transporta diferite informaţii despre participanţi, în cadrul unei sesiuni de transfer. [14]

- **Standardul RTCP** (Real time Tranfer Control Protocol) standardizează o modalitate de monitorizare respectiv corectare a erorilor de transmisie. [2]
- **Protocolul RTSP** (Real-Time Streaming Protocol) este un protocol de comunicare între un server multimedia şi un client.

- **Protocolul SIP** (Session Initiation Protocol) este un protocol de comunicare între un server multimedia si un client, este folosita de către serverele şi clienţii de Voice Over IP.

.3.3. Java Media Framework

Este dezvoltat de Sun Microsystems, special pentru a putea folosi fişiere de tip audio, video în aplicaţii dezvoltate folosind tehnologia Java. Oferă posibilitatea de a lucra cu date capturate de la diferite dispozitive ataşate calculatorului, având implementate clase specializate în acest sens. Uşurează foarte mult munca celui ce dezvoltă astfel de aplicaţii, framework-ul permiţând chiar transmiterea la distanţă a datelor. Desigur apare problema calităţii, deoarece sunt suportate doar anumite formate standard nu şi cele "high-quality" sau cele ce necesita lăţime mare de banda, acestea fiind destinate aplicaţiilor locale.

Java Media Framework este un API (Application Programming Interface) folosit pentru a include media de timp real în aplicaţii Java. Permite capturarea şi salvarea datelor de tip media, controlul procesării pe timpul vizionării, o procesare în funcţie de nevoi a datelor şi trimiterea, respectiv primirea, datelor de tip media de la un utilizator de la distanţă, folosind protocolul RTP. JMF oferă o arhitectură şi un protocol de mesaje pentru a realiza capturarea, procesarea, primirea şi transmiterea de date media în timp real. Este proiectat sa suporte tipuri de date media, cum ar fi AIFF, AU, AVI, GSM, MIDI, MPEG, QuickTime, RMF, and WAV.

JMF foloseşte acelaşi model de bază ca în figura 3.4. Un obiect *"data source"* va încapsula datele de tip media capturate sau primite, precum o caseta video, şi un obiect *"player"* sau *"processor"* va oferi procesarea şi mecanismul de control, precum un aparat video. Vizionarea şi capturarea audio/video folosind JMF necesită dispozitive precum microfoane, camere video, boxe şi monitoare.

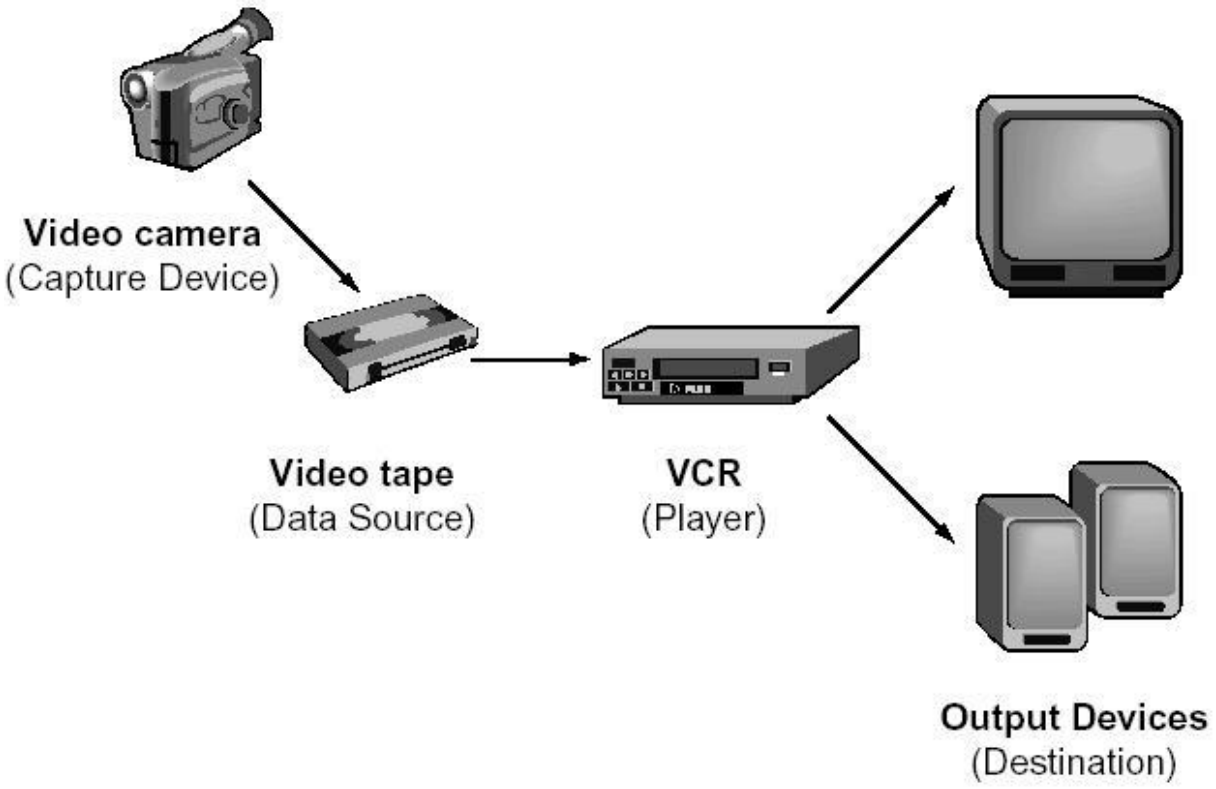

Figure 6 - Model de baza JMF

"*Data source"* şi *"Player"* sunt părţi ale nivelului superior al JMF, dar este oferit şi un nivel de jos ce suportă integrarea componentelor şi extensiilor procesate. Acest nivel oferă dezvoltatorilor de aplicaţii bazate pe media, audio/video, un API uşor de folosit flexibil şi extensibil în funcţie de cerinţele problemei. Pot fi implementate diferite decodificatoare sau codificatoare pentru diferite formate de date.

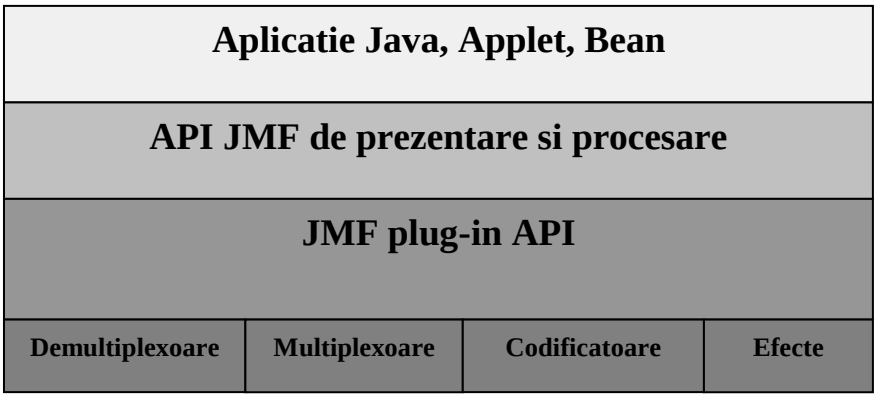

Figura 7 - Arhitectura JMF

Arhitectura de mai sus reprezintă o structurare pe nivele a ceea ce reprezintă acest API şi folosirea lui.

În continuare voi face o descriere pe scurt a claselor din JMF folosite în sistemul de supraveghere prezentat prin lucrarea de faţă:

- **MediaLocator** – identifică locaţia sursei de date, precum un URL.
- **SessionManager** – coordonează sesiunile RTP ţinerea în evidenţă a participanţilor şi a datelor ce sunt trimise într-o sesiune.
- **Manager** – clasa care face managementul general, creează obiectele *DataSource, Player, Processor*
- **Player** – Proceseaza date media de intrare de la o sursă de date şi îl livrează la un moment precis de timp. De asemenea oferă utilizatorului butoane de control standard, cum ar fi play, pause, etc.
- **Processor** – este un tip specializat de Player care oferă posibilitatea controlului procesării unor date media de intrare. Poate să trimită datele spre un dispozitiv de prezentare sau spre un alt obiect *DataSource*.
- **DataSource** – transmite datele media de intrare spre un *Player* sau *Processor*.
- **DataSink** – citeşte datele media de la o sursa de date, *DataSource*, şi le transmite la diferite destinaţii (fişier, reţea, RTP broadcaster, etc.).[9]

.4. Aplicaţii distribuite multinivel

Prin definiţie aplicaţiile distribuite au un grad de complexitate mai mare decât aplicaţiile locale, fiind necesar un "protocol" de comunicare, înţelegere, între componentele sau modulele ce formează sistemul distribuit. Cum la baza fiecărui produs software stă un proces software, acest proces va fi cu atât mai uşor de organizat în timp cu cât sistemul este mai modularizat.

A apărut în ingineria sofware nevoia diviziunii planurilor de activitate. Fiecare echipa de programatori este împărţită în mai multe sub-echipe, fiecare dintre acestea urmând să acţioneze asupra unui nivel al aplicaţiei. S-au definit trei nivele de acţiune: nivelul prezentare, nivelul logicii de business, nivelul integrării cu BD. La fiecare nivel se poate acţiona independent, în paralel până la un punct, legătura intre nivele realizându-se ulterior.

Prezentare - interfata utilizator

Aplicatie - functionalitatea aplicatiei

Date - date manipulate de client prin intermediul componentelor aplicatiei

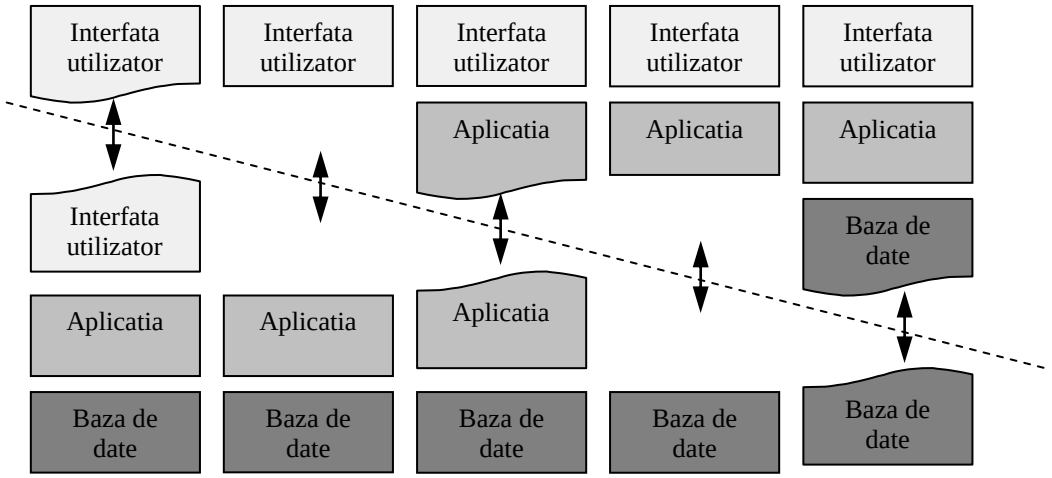

Figura 8 - Arhitecturi multinivel

.4.1. Nivelul prezentare

Nivelul prezentare reprezintă nivelul imediat al utilizatorului, interfaţa cu utilizatorul. La acest nivel de proiectare trebuie luate în considerare nevoile celui ce va avea contact direct cu aplicaţia. Prima nota pentru calitatea produsului este dată de acesta.

Interfaţa grafică trebuie sa respecte câteva reguli de bază: trebuie să fie uşor de utilizat, deci de înţeles – să fie explicită, trebuie să fie plăcută ochiului, obiectele conţinute să aibă nume sugestive care să descrie acţiunile ce vor avea loc odată cu utilizarea lor. Toate aceste calităţi dau un plus de apreciere din partea utilizatorului.

.4.2. Nivelul logicii de business

Nivelul business al unei aplicaţii distribuite se poate contura în jurul funcţionalităţii date de prelucrarea locală de date, la nivel de modul, de prelucrarea de cereri de la client către server, de logica folosirii obiectelor la distanţă.

Prelucrarea locală, la nivel de modul a datelor, se reduce la preluarea datelor utilizator şi procesarea acestora la nivel local, daca este posibil, iar daca este necesar, realizarea unei cereri către un server.

Modulul server va trebui să poată acţiona pe toate cele trei planuri de prelucrare a datelor amintite mai sus. În cazul în care are o interfaţă grafică, de obicei destinată unui administrator, va trebui să prelucreze datele administrator. Funcţionalitatea de server implică trimiterea de răspunsuri la cereri, răspunsuri care pot fi precedate de autentificări ale utilizatorilor ce au făcut cererea, dar şi tratarea de apel a obiectelor la distanţă.

.4.3. Nivelul integrării cu BD

.4.3.1. Generalităţi

S-a estimat că jumătate dintre aplicaţiile software ce sunt produse implică operaţii client/server, deci distribuire, şi mare parte din ele au legătură cu extragerea datelor din baze de date. Diferite companii s-au întrecut de-alungul timpului în proiectarea unor SGBD-uri eficiente, concluzia a fost aceea folosirii bazelor de date relaţionale, dar chiar şi între acestea existau diferenţe de la un producător la altul.

- **SQL (Structured Query Language)**

În aplicaţiile ce lucrează cu bazele de date se pune un mare accent pe operaţiile de memorare şi regăsire, efectuate asupra unor volume mari de date. Principala operaţie este aceea de regăsire, practic o baza de date este creată spre a fi interogată. Astfel este foarte important ca răspunsul sa fie unul rapid la orice interogare. Timpul de răspuns depinde desigur de structura bazei de date, de serverul de baze de date cu care se lucrează şi de driver-ul de acces la baza de date.

Limbajul SQL este un limbaj de interogare a SGBD-urilor relaţionale dezvoltat de IBM, pentru SGBD-ul propriu System R. Este disponibil şi sub alte SGBD-uri relaţionale ca ORACLE, MSSQL, MySQL. ([6], [7])

“Activitatea de organizare şi prelucrare a datelor a avut în ultimii ani o evoluţie determinată de doi factori esenţiali şi contradictorii care ar putea fi desemnaţi prin următorii termeni: necesităţi şi posibilităţi.” [6]

Necesităţile au fost mereu în creştere ceea ce a determinat o continuă evoluţie şi în cadrul posibilităţilor. Mereu a existat necesitatea accesului cât mai rapid la datele stocate.

În această continuă evoluţie au existat patru etape mari: prima etapă a evoluţiei tehnicii de organizare şi prelucrare a datelor, a doua etapa este marcată de separarea dintre structura logică de date şi structura fizică, etapa a treia este determinată de apariţia fişierelor integrate, iar a patra etapă este considerată etapa bazelor de date propriu-zise. S-a adoptat o arhitectură pe trei nivele a unei baze de date: nivelul intern (baza de date fizică), nivelul conceptual (modelul conceptual, schema conceptuală), nivelul extern (modelul extern, sunschema, vederea).

În timp au fost propuse mai multe modele de baze de date dintre care se disting: modelul ierarhic, modelul reţea şi modelul relaţional, care a devenit cel mai folosit.

S-a conturat noţiunea de sistem de gestiune a bazei de date – SGBD, sau DBMS (*Data Base Management System*), ca fiind întregul ansamblu software care tratează toate cererile de acces la baza de date, din partea diferiţilor utilizatori. Au apărut diferite companii strict specializate pe această ramură a SGBD-urilor, şi care au reuşit să ajungă la un nivel de dezvoltare, în această direcţie, care satisface orice nevoie în privinţa accesului, protejării, procesării, interogării datelor. A apărut problema standardizării limbajului de interogare a bazei de date şi astfel s-a născut şi standardul de facto în acest domeniu, Structured Query Language, SQL. Acesta are compatibilitate cu bazele de date produse de toţi mari investitori din acest domeniu.

Tabele, Proceduri stocate, Vederi

Orice bază de date relaţională are la bază conceptul de tabelă, reprezentând nivelul de deasupra datei efective. Orice tabelă poate să aibă mai multe câmpuri care împreună formează o tuplă, o linie de tabela. Bazele de date relaţionale, şi nu numai, permit identificarea unică a tuplelor, prin setarea de chei. Orice interogare făcută pe baza de date se va face relativ la una sau mai multe tabele. Acestea pot fi interdependente prin setarea unor legături între ele, asigurând astfel consistenţa datelor.

Vederile sunt reprezentate practic tot de tabele, tabele virtuale ce au în spate o interogare. Apariţia lor a fost necesară pentru a aduce un plus de operabilitate bazelor de date. Vederile prezintă datele ce sunt interesante la un moment dat pentru un anumit utilizator. Conturează practic un nou nivel de securitate, utilizatorul având acces doar la vedere nu la datele efective, el va primi doar ceea ce doreşte administratorul bazei de date. Se poate astfel face o validare înaintea stocării efective a datelor utilizator în baza de date, trebuind sa fie respectate anumite restricţii în cazul operaţiilor de ştergere, inserare sau actualizare.

Atunci când se lucrează cu o bază de date într-o aplicaţie, se poate vorbi de două metode pentru stocarea şi executarea programelor de acces la baza de date. Programele pot fi memorate local la nivelul aplicaţiei care trimite comenzi către serverul de baze de date şi prelucrează rezultatele returnate. O a doua opţiune ar fi aceea de a realiza proceduri stocate care să fie apelate din aplicaţii, iar apoi să se prelucreze rezultatele returnate de acestea. ([6], [7])

.4.3.2. Conectivitate Java-Baze de date

O mare parte dintre aplicaţiile software sunt distribuite, iar dintre acestea majoritatea au de a face cu baze de date. Până la apariţia SQL a existat problemă interogării acestora, odată rezolvată această problemă a apărut problema utilizării lor din aplicaţii software, deoarece nu există un "standard" care să dea portabilitate aplicaţiilor ce foloseau baze de date.

JDBC

Există un limbaj de interogare standard pentru bazele de date, reprezentat de SQL, dar existau probleme mari la realizarea conexiunii la baza de date, datorită diferenţelor ce existau între standardele impuse de marile companii în domeniu. Chiar dacă nu s-a ajuns la o mapare exactă a tipurilor impuse de bazele de date cu acelea existente în Java, diferenţele sunt rezolvate prin folosirea doar a tipurilor standard.

"Java DatBase Connectivity" este proiectat sa fie independent de platforma, aşa că dispare grija bazei de date folosite, în timpul dezvoltării de software. Oricum există posibilitatea de a folosii implementări specifice unui producător, deci nu există restricţii de la a face ceea ce se doreşte. Cele mai multe tipuri de baze de date suportă tipurile generice ale SQL, deci este indicată folosirea în aplicaţii a acestora şi nu a celor specifice unui producător. Problema portabilitaţii apare în momentul în care un utilizator încearcă citirea bazei de date cu surse necunoscute.

JDBC oferă un driver de control care ştie să comunice cu orice obiect driver necesar pentru a interoga baza de date. Aşadar dacă o aplicaţie necesită conectarea la mai multe baze de date diferite, din punctul de vedere al producătorului, vor fi necesare obiecte driver pentru fiecare dintre ele. Obiectul driver se va înregistra la driverul de control la momentul încărcării, ce se poate forţa prin apelul: *Class.forName().*

Pentru a se deschide o conexiune la o baza de date este necesară construirea unei adrese URL a acesteia. De exemplu: *jdbc:microsoft:sqlserver://serverName:serverPort*. De asemenea este necesară autentificarea, adică un nume de autentificare şi o parolă, practic o conexiune se va obţine folosind toţi aceşti trei factori enumeraţi.

Connection c = DriverManager.getConnection(dbUrl, user, password);

Lucrul cu acest pachet este relativ simplu, partea mai complicată poate fi considerată aceea a utilizării obiectului driver, deoarece fiecare are particularităţile sale. Problemele pot să apară în a-l face să se încarce cum trebuie şi în a seta baza de date folosind instrumentul de administrare a acesteia.[5]

DAO

DAO este un şablon de implementare ce abstractizează şi încapsulează accesul la sursele de date, de orice fel ar fi acestea. Implementează mecanismul de acces necesar folosirii sursei de date.

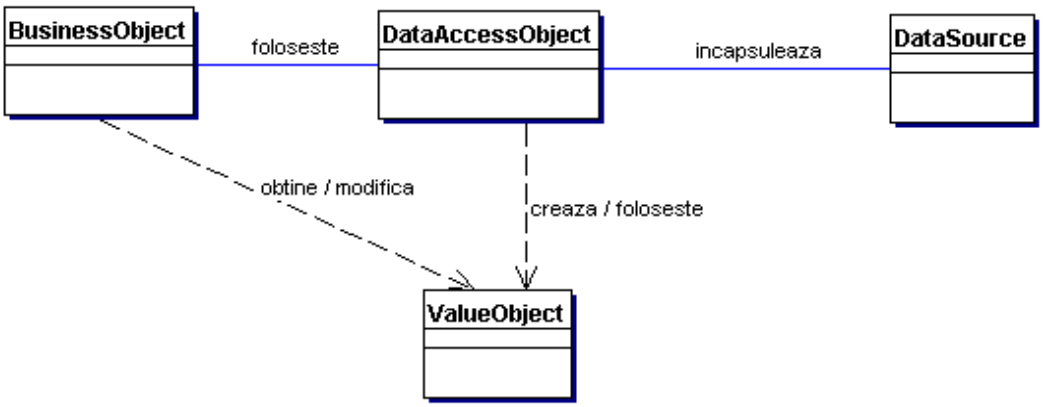

Figura 9 - Data Access Object

Participanţi:

1. BusinessObject – reprezintă clientul ce face acces la date.

Un BusinessObject poate fi implementat ca şi un Session Bean, Entity Bean, sau alt obiect Java .

2. DataAccessObject

DataAccessObject este obiectul de baza al acestui şablon. El abstractizează implementarea accesului la sursa de date oferind o interfaţă de acces transparent pentru

BusinessObject. BusinessObject va delega încărcarea şi stocarea de date unei entităţi DataAccessObject.

3. DataSource

Reprezintă o implementare a sursei de date. O sursă de date poate fi un SGBDR, SGDDOO, fisiere XML, fişiere tabelare, etc.

4. ValueObject

Value object este folosit ca entitate purtătoare de date. DataAccessObject va folosi un ValueObject pentru a întoarce date către BusinessObject. Entitatea DataAccessObject va primi de asemenea date încapsulate într-o entitate ValueObject pentru a actualiza sursa de date.

Obiectul business, din figura, care se bazează pe DAO foloseşte doar o interfaţa expusă pentru clienţi, detaliile de implementare în ceea ce priveşte lucrul cu sursa de date sunt încapsulate şi ascunse clienţilor. Deoarece interfaţa cu clienţii nu se schimbă când implementarea sursei de date se schimbă, şablonul DAO oferă posibilitatea adaptării la diferite scheme de salvare de date, fără să afecteze clienţii sau componentele bussiness.

Şablonul DAO poate fi extrem de flexibil prin adoptarea şabloanelor *"Abstract Factory"* şi *"Factory Method*". Când sursa de date este un subiect ce nu se va schimba de la o implementare la alta aceasta se va adopta Factory Method pentru a produce entităţile DAO necesare în aplicaţie.[15]

2. Calitate in inginerie software

Procesul software reprezintă un şablon pentru crearea produsului software. Sunt cunoscute câteva modele de procese: modelul cascada, dezvoltare prin prototipizare, modele evolutive, dezvoltare orientată pe componente, dezvoltare prin metode formale. În funcţie de cerinţele de început ale proiectului, de cât de explicite sunt şi de posibilitatea schimbării acestor cerinţe se poate lua o decizie în privinţa modelului ce va fi folosit.

Pentru a urmării factorii de calitate, descrişi în obiectivele nefuncţionale, trebuie urmărite arhitecturi şi modele, atât în procesul software în general, dar şi la nivel de implementare efectivă a aplicaţiei.

2.1. Proces software. Modelul evolutiv

Chiar dacă par ambigue, cerinţele unui proiect reprezentat de un sistem de supraveghere sunt foarte clare, deoarece este bine cunoscută funcţionalitatea ce trebuie să o aibă la sfârşit un astfel de sistem. Cu toate acestea un astfel de sistem trebuie sa ţina pasul cu modificările ce apar în partea de hardware folosit şi să poată răspunde unor cerinţe suplimentare din partea utilizatorului. Astfel decizia aplicării unui model evolutiv pare a fi cea mai bună. Dintre acestea cel care suportă cel mai bine condiţiile unui sistem software de supraveghere este dezvoltarea incrementală, deoarece este clară apariţia unei noi versiuni în cazul modificărilor unor condiţii impuse hardware, vechea versiune rămânând funcţională, dar fiind limitată. De asemenea suprapunerea în timp a realizării diferitelor versiuni este necesară, versiunea mai slabă nu se va abandona deoarece nu este complet inutilă.

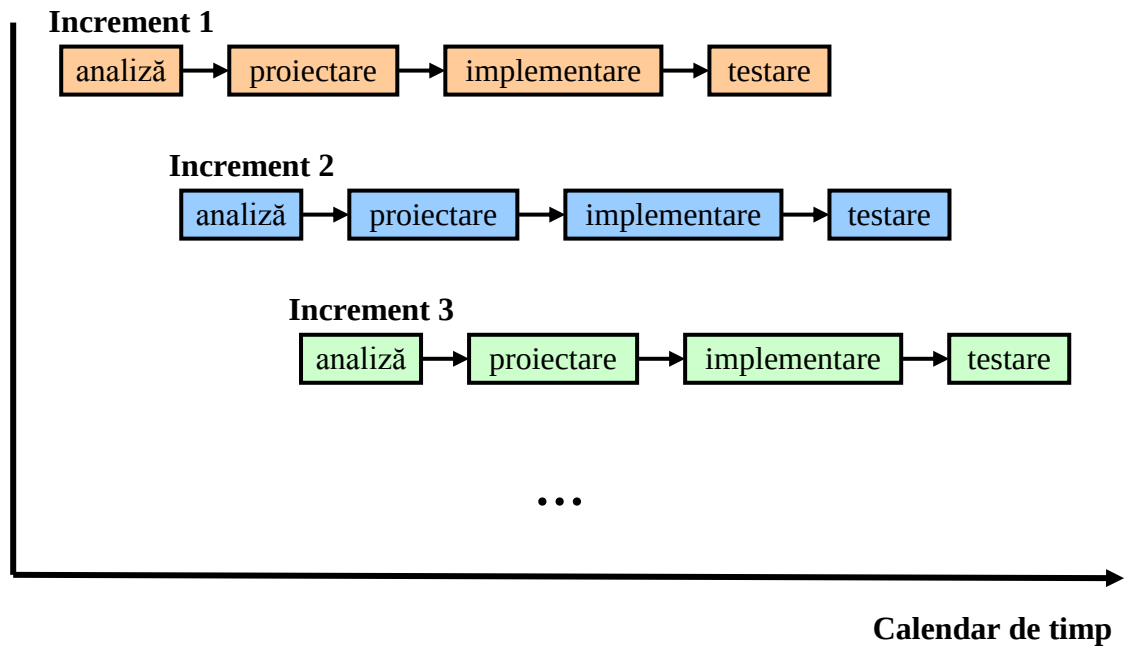

Figura 10 - Modelul evolutiv

“Succesul abordării incrementale depinde de modul în care este realizată maparea cerinţelor la incremente.” Desigur în cazul unor cerinţe exprimate clar munca va fi mult mai simplă şi succesul garantat. Fiecare increment trebuie să fie relativ mic, iar cerinţele critice ale produsului trebuie tratate încă din primele incremente. Astfel se vor reduce riscurile şi se vor evita aglomerările din fazele de sfârşit.

Chiar dacă în figura de mai sus fiecare increment apare dezvoltat într-un proces linear, dacă la un moment dat specificaţiile nu sunt clare sau ar putea suferii modificări ulterioare în realizarea acelui increment se pot folosi alte modele de proces software, cum ar fi explorare sau prototipizare.

Având în vedere că dezvoltarea software acţionează în cadrul unor constrângeri de timp şi mai ales de buget, devine absolut necesară o programare riguroasă a procesului software. Consecinţa unei bune organizări în timp aduce un plus de calitate produsului, deoarece echipa de dezvoltatori nu va fi nevoită să renunţe la vreunul din paşii propuşi în dezvoltare. [8]

2.2. Arhitectura MVC

MVC reprezintă o arhitectura folosită în special în aplicaţiile în care se realizează o interfaţa grafică cu utilizatorul. Reprezintă o soluţie pentru a împărţi aplicaţia, sau doar o parte a ei, în trei părţi: modelul, vederea şi controlerul. A fost dezvoltată pentru a mapa rolul binecunoscutei arhitecturi input->processing->output pe aplicaţiile ce implică GUI (Graphical User Interface). Modelul are sarcina interfeţei funcţionale a sistemului, răspunzând la diferite cereri, vederea este dată de interfaţa grafică utilizator şi chiar liniile de text de la consola, practic ceea ce se vede, iar controlerul are rolul interpretării acţiunilor utilizatorului şi comandării modelului şi/sau vederii spre a-şi schimba starea. [16]

În arhitectura MVC intrările utilizatorului, modelul lumii exterioare, şi reacţia vizuală la utilizator sunt separate explicit şi controlate de trei tipuri de obiecte, fiecare specializat pentru partea sa de control. Vederea controlează ieşirea grafică sau în formă de text pe monitor a aplicaţiei. Controlerul interpretează acţiunile de mouse sau tastatură ale utilizatorului, comandând modelul şi/sau vederea spre a-şi schimba starea în funcţie de rezultatul interpretării. Modelul controlează comportamentul şi datele domeniului aplicaţiei, răspunde la cereri privind starea sa, de obicei de la vedere, şi răspunde la instrucţiuni de schimbare a stării, de obicei de la controler. Separarea formală a acestor trei task-uri este o noţiune importantă, în particular, prevăzută la Smalltalk-80, unde comportamentul de bază poate fi încapsulat în obiecte abstracte: Vedere, Controler, Model şi Obiect. Comportamentul MVC este moştenit, adăugat şi modificat pentru a oferi sisteme flexibile şi puternice.

Pentru a putea folosi această arhitectură trebuie mai întâi înţeleasă diviziunea pe care MVC o face, iar apoi felul în care comunică cele trei părţi între ele sau cu alte vederi sau

controlere.

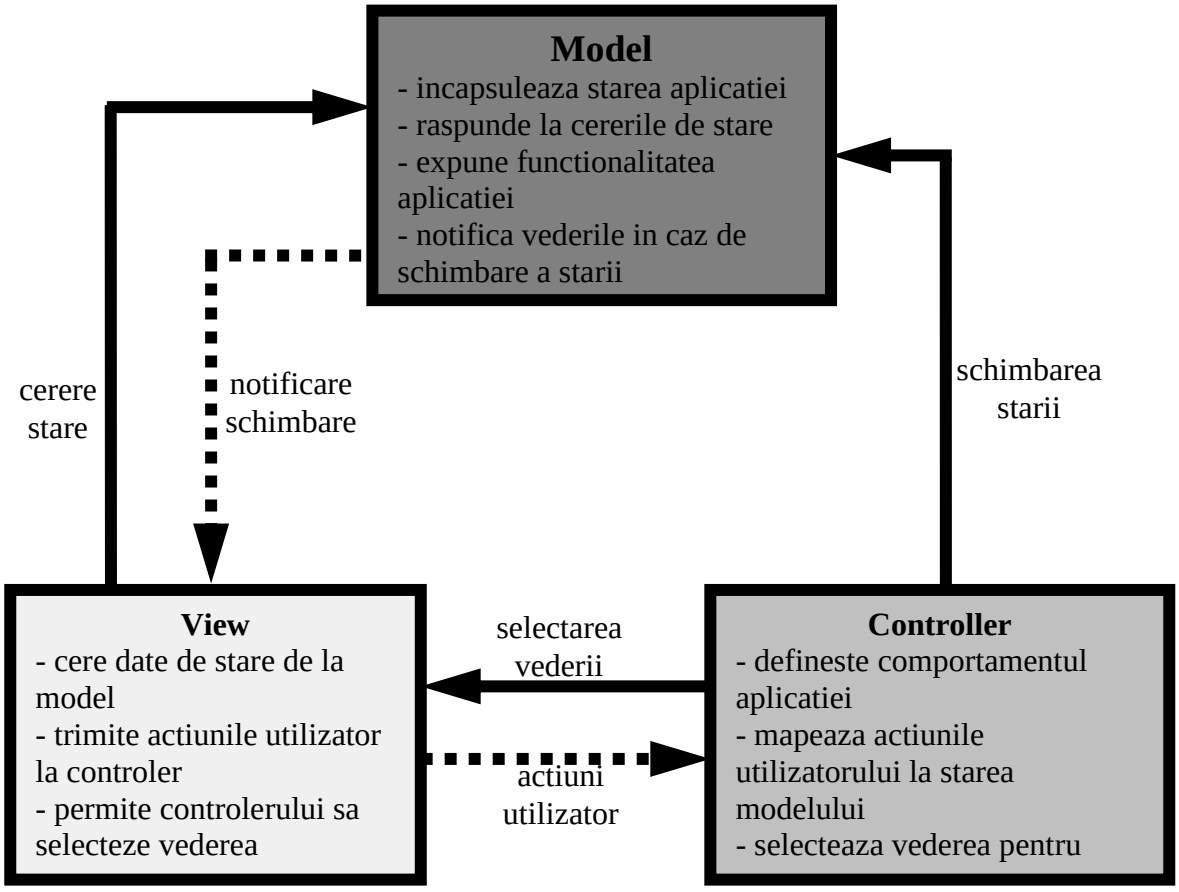

Figura 11 - Arhitectura MVC

Într-o aplicaţie distribuită, în care modulele trebuie să comunice între ele, împărţirea pe care o oferă utilizarea arhitecturii MVC este foarte utilă deoarece defineşte foarte clar părţile de componenta, modul, care comunică. Se poate crea o interacţiune constructivă între interfaţa grafică, a aplicaţiei de la distanţă, şi aplicaţia locală, datorită posibilităţii de transfer a datelor în aplicaţie pe care o dă o construcţie pe structura MVC.

III. Specificaţiile si arhitectura sistemului

Sistemul S.V.L.W. (Supraveghere Video in LAN sau WAN) oferă posibilitatea de a realiza supraveghere video a unei incinte, dar şi de a viziona fişiere multimedia capturate în alte momente de timp, decât cel actual. Supravegherea se face folosind un dispozitiv de captură conectat la un calculator, acestea reprezentând mijloacele fixe de la locul supravegheat, şi un calculator la distanţă reprezentând obiectul de control al imaginilor capturate. Posibilitatea înregistrării şi vizionării ulterioare este dată de o a treia componentă, serverul de date.

1. Funcţiile sistemului

După o privire de ansamblu asupra domeniului multimedia şi după testări, în scopul descoperirii funcţionalităţii altor produse, am reuşit formularea câtorva funcţii concrete pe care sistemul de supraveghere S.V.L.W. ar trebui să le îndeplinească. Punctul de pornire şi cerinţele de bază ale acestuia sunt funcţiile de bază ale unui sistem de supraveghere de orice gen. Desigur toate aceste funcţii pot fi mapate peste obiectivele funcţionale descrise la începutul lucrării.

a. Captura video

O prima cerinţă a unui sistem ce are ca scop supravegherea la distanţă este capturarea de imagini reprezentând activitatea la un moment dat de timp. Pentru a realiza aceasta este nevoie de un dispozitiv de captura instalat la un calculator gazdă. Este necesară instalarea unui driver pentru dispozitivul respectiv, pentru a se reuşi comunicarea. Este de la sine înţeleasă necesitatea posibilităţilor de conectare a respectivelor dispozitive, majoritatea având ca şi interfaţa de conectare portul USB.

Vizualizarea imaginilor capturate pe calculatorul gazdă al dispozitivului, poate reprezenta o funcţie a sistemului, aceasta realizându-se cu ajutorul unei aplicaţii de captură. O astfel de aplicaţie este relativ uşor de realizat cu ajutorul framework-ului JMF, de la Sun Microsystems – modulul CamProcessor.

b. Transferul de date multimedia

Odată datele capturate, ele trebuie vizualizate şi analizate, desigur nu este îndeajuns o vizualizare a lor locală, la calculatorul ce are conectat dispozitivul de captură. Astfel este necesară transmiterea datelor spre un alt calculator din reţeaua locală sau din

internet. Datele de timp real reprezintă un tip de date special şi necesită un protocol de transfer special. Este necesară fluenţa transmisiunii şi nu siguranţa şi integritatea datelor, de aceea pentru astfel de conexiuni se folosesc protocoale speciale – RTP/RTCP.

c. Vizualizarea date de timp real la distanta

Odată primite datele de timp real trebuie vizualizate. Şi acest capăt de conexiune este necesară o aplicaţie ce poate să lucreze cu astfel de date. De asemenea nu este acceptabil ca vizualizarea sa înceapă după terminarea transferului, deci o stocare prealabilă. Datele vor fi vizualizate odată cu sosirea lor.

Desigur în tot acest proces apare o întârziere dată de transferul prin reţea, care va fi cu atât mai mare cu cât "distanţa" dintre cele doua capete de conexiune este mai mare. Aceste întârzieri nu sunt resimţite de utilizator, deoarece principala sa preocupare va fi calitatea datelor, dată de fluenţa datelor multimedia.

d. Înregistrarea date de timp real

Orice sistem de supraveghere trebuie să permită înregistrarea datelor între anumite perioade de timp, pentru o eventuală reanalizare ulterioară. Decizia de înregistrare este luată de utilizatorul ce supraveghează activitatea, dar este făcută de aplicaţia de captură, pentru a oferi o calitate cât mai bună.

e. Transferul datelor înregistrate

Datele multimedia înregistrate trebuie transferate de pe calculatorul pe care a fost făcută înregistrarea, astfel riscându-se supraîncărcarea ca la un moment dat să se ajungă în incapacitatea de a mai înregistra din cauza lipsei de spaţiu. Astfel datele înregistrate sunt transmise spre un calculator server pe care va fi asigurat spaţiul necesar stocării.

f. Gestionarea datelor înregistrate

Înregistrarea datelor pe server fiind realizată, pentru o mai bună gestiune, vor fi înregistrate într-o bază de date, alăturându-li-se caracteristici suplimentare date de poziţionarea aparatului ce le-a capturat.

g. Gestionarea datelor adiţionale

În acest cadru sunt incluse mai multe funcţionalităţi ale sistemului:

- *autentificarea utilizatorilor*

Utilizatorii, cu titlu de administratori, ai aplicaţiei sunt identificaţi, in momentul în care încearcă să modifice sau să adauge date care ţin de configurarea administrativă a sistemului, cu ajutorul unui număr de înregistrare şi a unei parole. Accesul la date şi posibilitatea efectuării de operaţii pe aceste date este permis doar după validarea acestor informaţii, păstrate într-o tabelă separată a bazei de date cu care se lucrează.

- *obţinerea şi managementul informaţiilor utile*

Reprezintă funcţia de interfaţa între utilizatorul, administrator de sistem, şi baza de date din spatele sistemului. Administratorul are posibilitatea completării, modificării sau ştergerii datelor, pentru a reconfigura sistemul.

- adăugarea, ştergerea zonelor arealului supravegheat
- adăugarea, ştergerea locaţiilor dispozitivelor de captură
- adăugarea, ştergerea locaţiilor aplicaţiilor de analiză
- localizarea dispozitivelor de captură

2. Cazuri de utilizare

Analizând cerinţele şi funcţiile ce trebuie să le îndeplinească sistemul în varianta sa finală s-au identificat câteva cazuri de utilizare. Modularizarea aplicaţiei duce la existenţa a trei tipuri de utilizatori: spectator, supraveghetor şi administrator.

- **spectator**

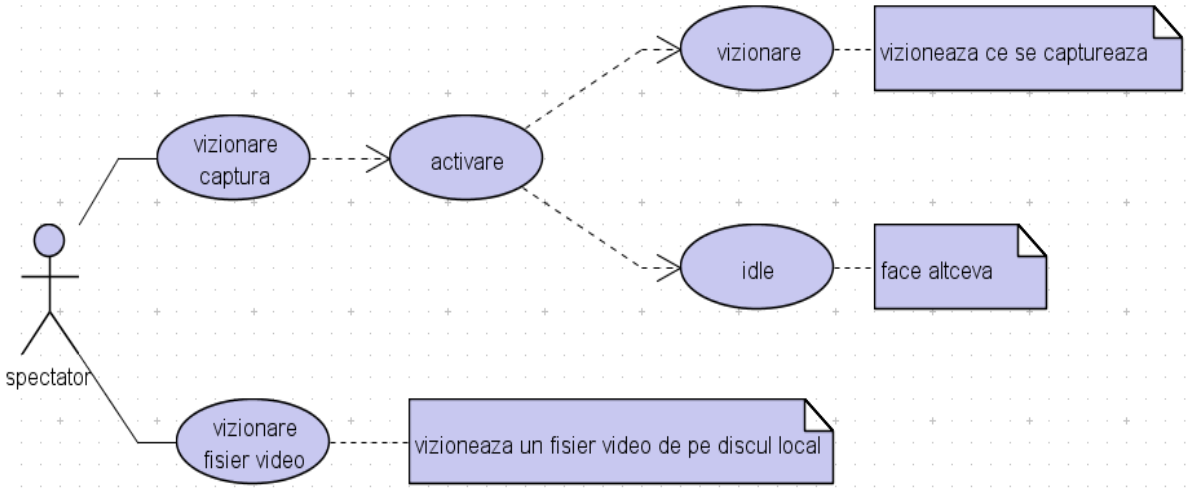

Figura 12 – Spectator-Use Case

Actorul, spectator, are două posibilităţi în funcţie de modul în care porneşte aplicaţia: poate să vizioneze, sau nu, ceea ce se capturează, poate să vizioneze un fişier video de pe discul său local.

Diagrama de activitate aduce specificaţii clare privind acţiunile şi deciziile utilizatorului în timpul folosirii aplicaţiei.

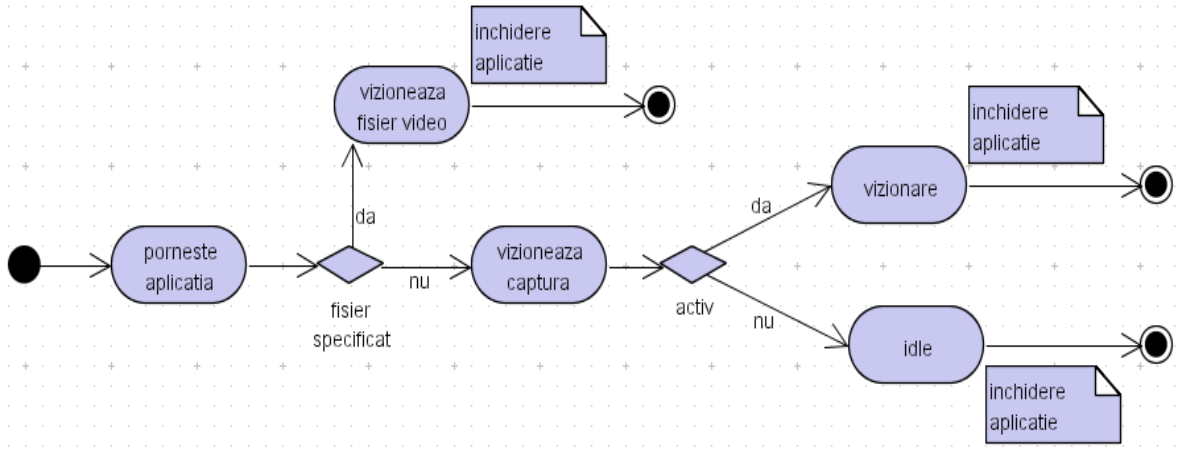

Figura 13 – Spectator-Diagrama de activitate

- **supraveghetor**

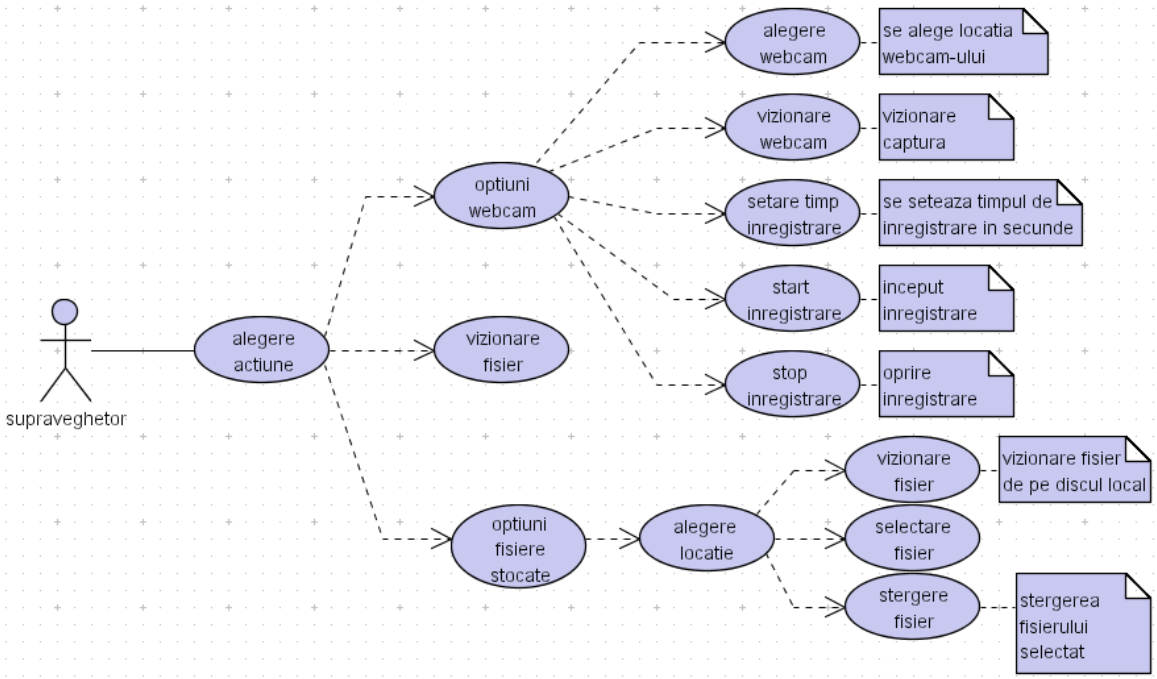

Figura 14 – Supraveghetor-Use Case

Actorul, supraveghetor, deţine controlul supravegherii, el fiind cel care analizează imaginile şi ia hotărâri de a înregistra sau şterge fişiere. Poate să schimbe în orice moment locaţia ce o supraveghează şi are posibilitatea să vizioneze un fişier de pe discul său local.

Activitatea utilizatorului este surprinsă în diagrama de activitate pentru cazul său de utilizare:

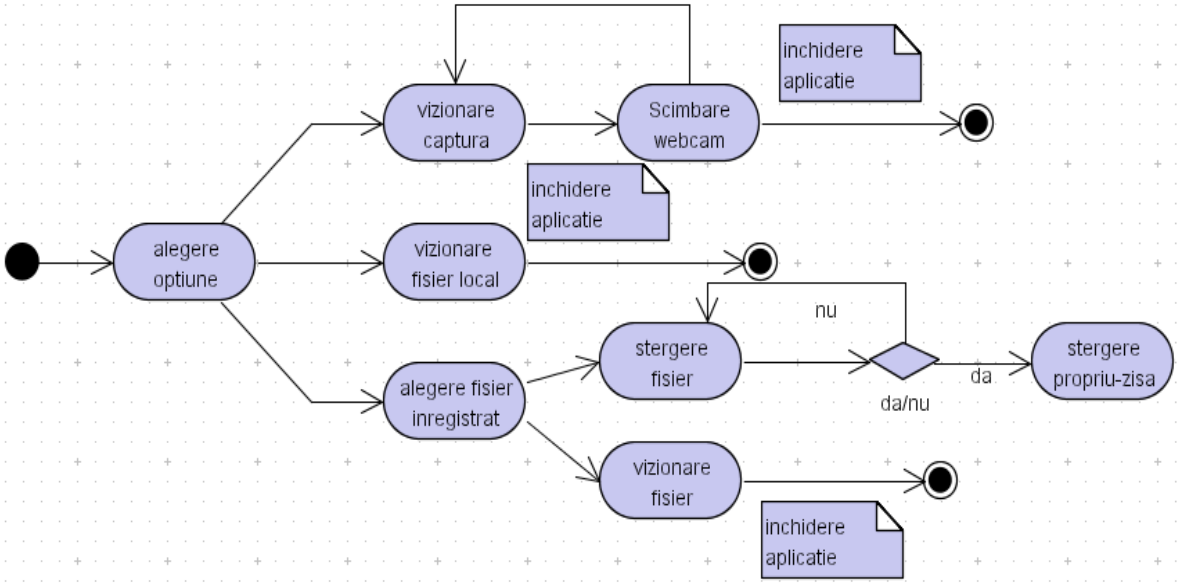

Figura 15 - Supraveghetor-Diagrama de activitate

- **administrator**

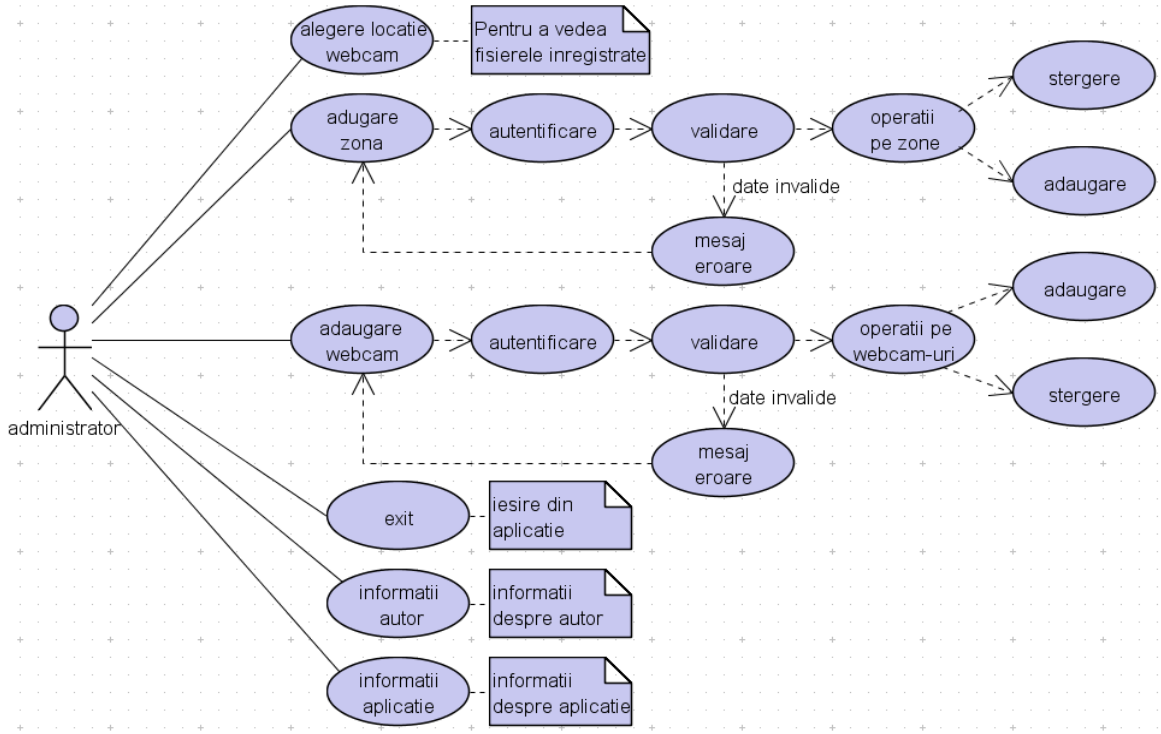

Figura 16 – Administrator-Use Case

Actorul, administrator, deţine mai mult controlul administrativ al aplicaţiei de supraveghere, decât un rol activ. Acţiunile sale nu sunt direct resimţite de ceilalţi utilizatori ai aplicaţiei. Administratorul are rolul de manager al resurselor pe care le foloseşte aplicaţia.

Adiacentă cazului de utilizare, diagrama de activitate prezintă acţiunile explicite ale utilizatorului.

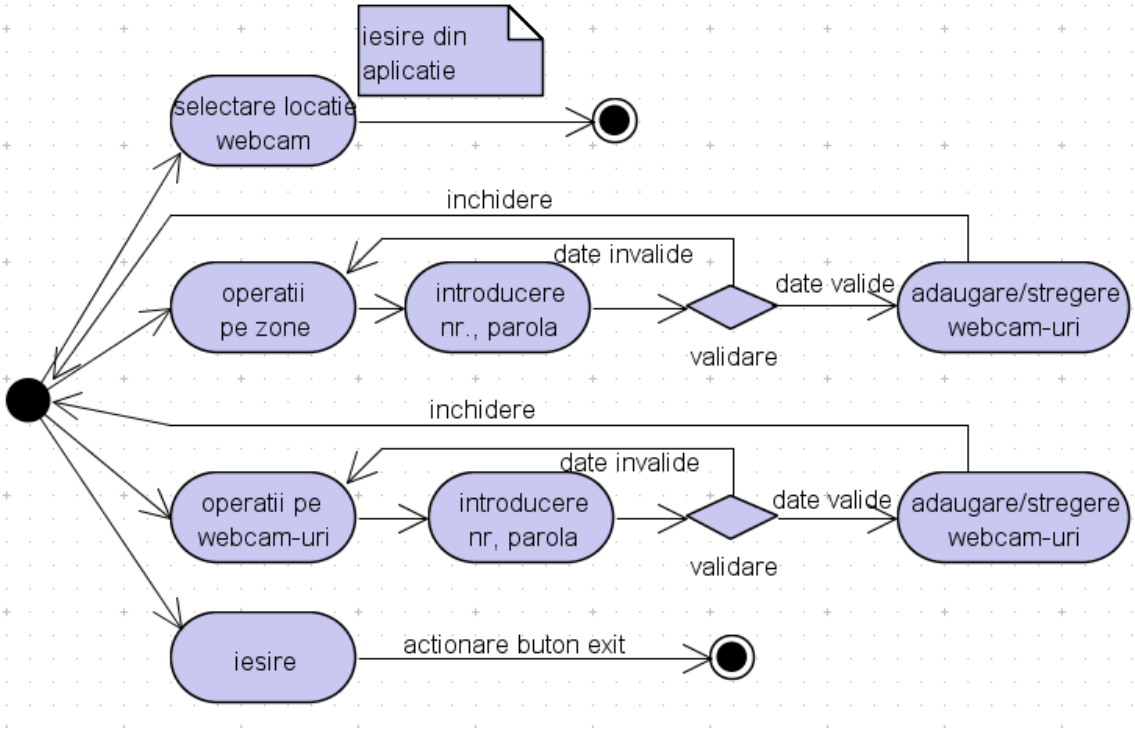

Figura 17 – Administrator-Diagrama de activitate

Cei trei utilizatori, reprezentaţi în diagrame ca actori, acţionează independent unul de celalalt, în afara acţiunilor de închidere a aplicaţiei orice altă acţiune nu contează pentru ceilalţi doi oricare ar fi ei.

3. Schema bloc a sistemului

Aplicaţia şi-a propus realizarea unui sistem software de supraveghere, distribuit, în mai multe module ancorate într-un middleware ce să permită apelare de metode şi obiecte la distanţă. Middleware-ul este oferit de ORB-ul ORBacus, de la Iona, şi în plus pentru servicii de apel la distanţă sunt integrate serviciile CORBA.

Sistemul este reprezentat de cele trei tipuri de module ce îl compun şi comunicaţia ce se realizează între ele:

- ***Modulul server de date***
  - înregistrări / deregistrări de obiecte de procesare şi controler
  - salvare fişiere multimedia
  - transmitere de fişiere de timp real anterior înregistrate
  - deţine conexiunea cu un server de baze de date
- ***Modulul de captură***
  - captura imaginii cu ajutorul unui dispozitiv de captură de tipul cameră video sau webcam
  - transmite datele de timp real capturate către un eventual controler
  - transmite fişierele înregistrate către serverul de date
- ***Modulul de control***
  - preia imagini de timp real actuale de la diferite procesoare de imagini
  - poate viziona imagini de timp real în prealabil înregistrate, pe care i le trimite serverul de date
  - controlează procesul de supraveghere

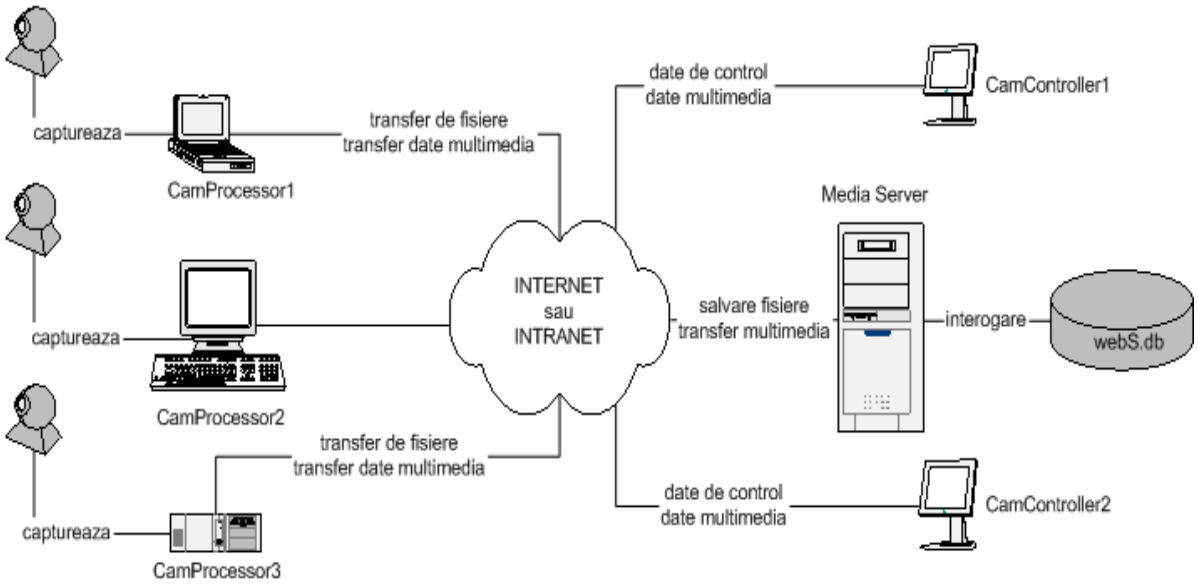

Figura 18 - Schema bloc a S.V.L.W.

Elementele ce formează sistemul va trebui să respecte câteva reguli, pentru buna funcţionare a aplicaţiei.

Modulul de captură trebuie să aibă ca şi gazda un calculator cu port USB, deoarece majoritatea dispozitivelor de captură se conectează folosind acest tip de port. Este necesar să

aibă legătură la reţea care să suporte cel puţin 500kbps, necesari transferului multimedia în condiţii optime de calitate până la un eventual modul de supraveghere.

Modulul server de date trebuie să fie pe un calculator cu spaţiu de stocare mare, pentru a putea memora fişierele înregistrate de utilizatorii activi. Trebuie să aibă o conexiune la un server de baze de date, ce găzduieşte baze de date a sistemului de supraveghere.

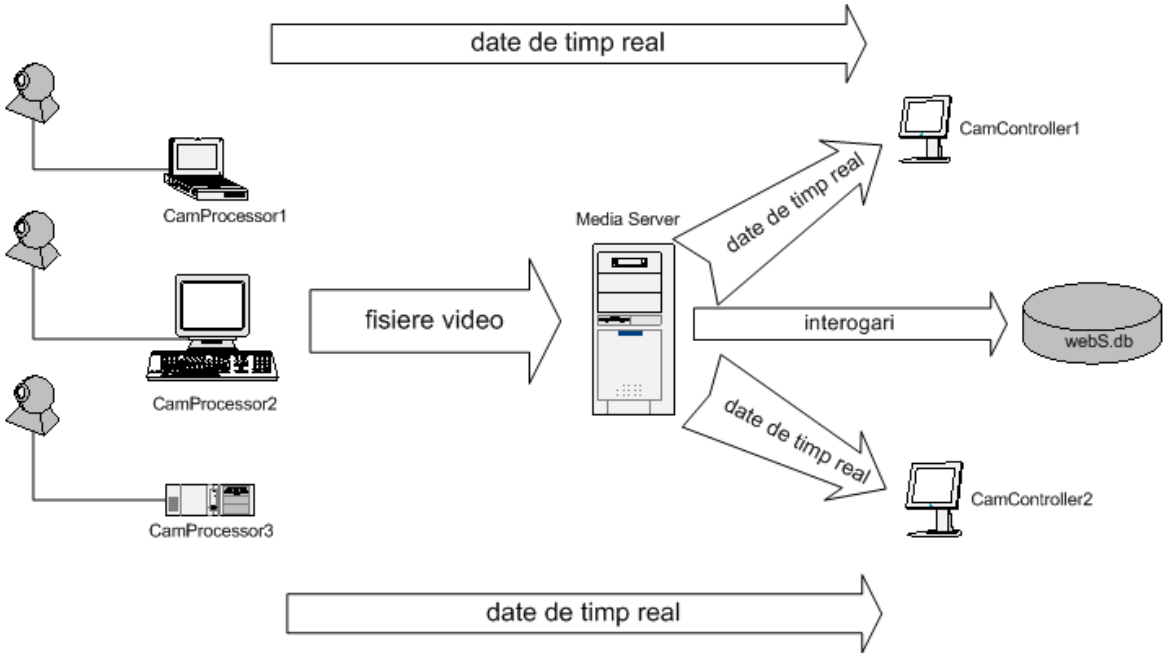

Figura 19 – Schema bloc

4. Arhitectura generală

Soluţia propusă are o arhitectură client/server, în care entităţile prezente în sistem devin, în funcţie de situaţie, client sau server. Entităţile sistemului sunt cele prezentate ca module în secţiunea anterioară.

Entitatea "server de date" este server-ul general al aplicaţiei, pe această maşină fiind pornit şi serviciul de nume CORBA, dar şi server-ul bazat pe socket pentru transferul fişierelor înregistrate de entităţile de captură. Server-ul tratează cereri de la 2 feluri de clienţi: clientul dat de entitatea de captură şi clientul dat de entitatea de control.

Serviciul de nume este realizat practic prin pornirea, pe o maşină gazdă, a unui ***server de nume*** suplimentar pentru satisfacerea serviciului. Acest server va realiza înregistrarea şi legarea numelui de referinţă. Serverul de nume este pornit pe un anume port.

Datele privind numele gazdei şi portul pe care este pornit serverul de nume trebuie cunoscute de toţi clienţii care vor folosi acest serviciu. Aşadar în momentul lansării modului de captură trebuie cunoscute aceste date, de asemenea la momentul lansării modulului de control.

Serverul de fişiere este un server socket pornit tot pe maşina gazda a server-ului de date pentru a putea fi colectate fişierele înregistrate de entităţile de captură. Datele ce trebuie cunoscute sunt aceleaşi ca în cazul server-ului de nume, adică numele gazdei şi portul pe care a fost pornit serverul. Odată pornit se vor putea face cereri de conectare şi odată realizată conexiunea se va începe transmiterea de fişiere într-un fir de execuţie separat. Transmiterea de fişiere înregistrate se face prin tehnologia socket cunoscută, două zone tampon: de intrare şi de ieşire. Zona tampon de intrare a clientului va fi practic zona tampon ieşire a server-ului, iar zona tampon de ieşire a clientului este zona tampon de intrare a serverului.

Server-ul de date realizează o conexiune la un server de baze de date care găzduieşte baza de date a aplicaţiei. Pentru a realiza această conexiune se foloseşte un DriverManager, furnizat de JDBC, şi trebuie cunoscut numele serverului, utilizatorul şi parola sa, necesare autentificării în cadrul serverului de baze de date. În această situaţie serverul general al aplicaţiei devine client al unui alt server, acela de baze de date. Server-ul de date va face interogări asupra server-ului de baze de date, pentru a obţine, adăuga sau schimba conţinutul tabelelor în baza de date.

Entitatea de captură este dată de modulul de captură care în sistem pare să aibă rolul cel mai important, deoarece fără el funcţionalitatea aplicaţiei ar avea cel mai mult de suferit. Captura imaginilor de la dispozitivul de captură se face cu ajutorul funcţiilor puse la dispoziţie de framework-ul JMF, iar la nivel mai jos de driver-ul dispozitivului respectiv şi librăriilor de funcţii ale sistemului de operare.

În figura de mai jos este prezentată schema de captură video în sistemul de operare Windows NT, până la nivelul driver-ului de dispozitiv. De aici controlul este preluat de JMF, care reuşeşte prin clasele care le pune la dispoziţie să gestioneze informaţia spre a-i da forma ce o are în final utilizatorul în faţa.

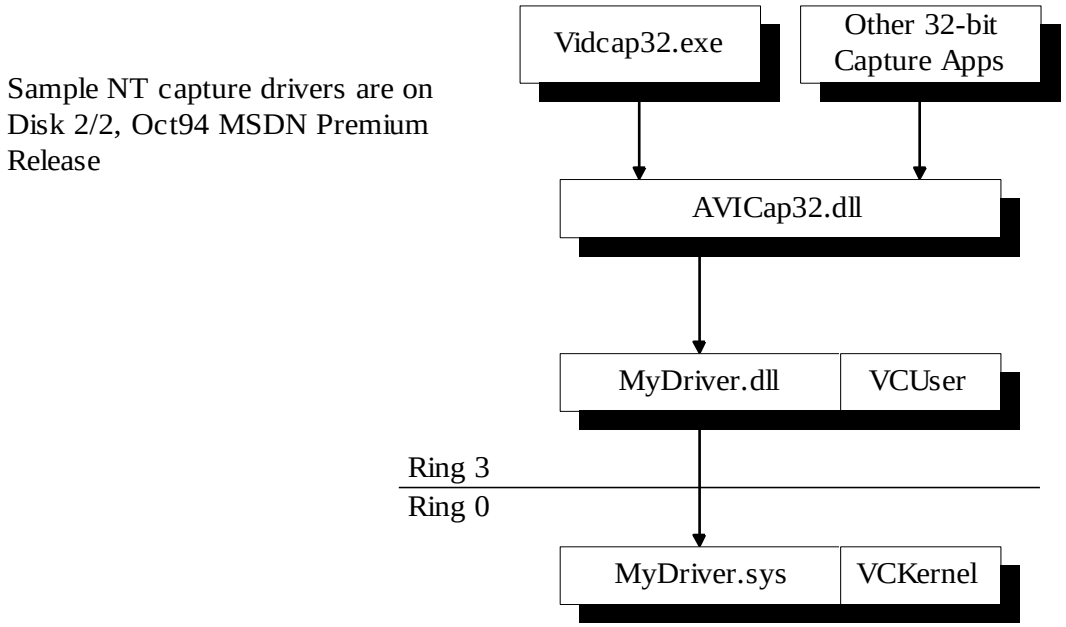

Figura 20 – Captura video

Pe lângă funcţia de captură entitatea mai are funcţia de "server" pentru entitatea de control, care va face cereri spre a-i trimite date de timp real în scopul controlului şi supravegherii. Practic în cadrul entităţii este realizat un obiect manager care va rezolva aceste cereri de transmisie a datelor de timp real.

La nivelul acestei entităţi se realizează salvarea pe disc a fişierului comandat de entitatea de control. Acest fişier va fi transferat pe server după terminarea înregistrării, după cum am descris mai sus.

Entitatea de control este înfăşurarea modulului de control. Funcţiile ce le acoperă sunt acelea ce dau rezultate finale, la acest nivel al aplicaţiei se iau deciziile de control, deciziile specifice acestui gen de sisteme. Entitatea de control este clientul "absolut" al aplicaţiei de supraveghere. Face cereri atât către modulul server de date, cât şi către modulul de captură, de-asemenea fiind client al server-ului de nume.

Server-ul de date primeşte cereri de la modulul de control pentru a-i transmite în timp real fişiere înregistrate anterior, tot din comanda dată de la modulul de control.

Entitatea de captură primeşte cererea de la modulul de control pentru a-i trimite datele de timp real ce le capturează, dar şi cererea de a începe înregistrarea datelor pentru un anumit timp precizat.

Figura 21 – Arhitectura aplicatiei

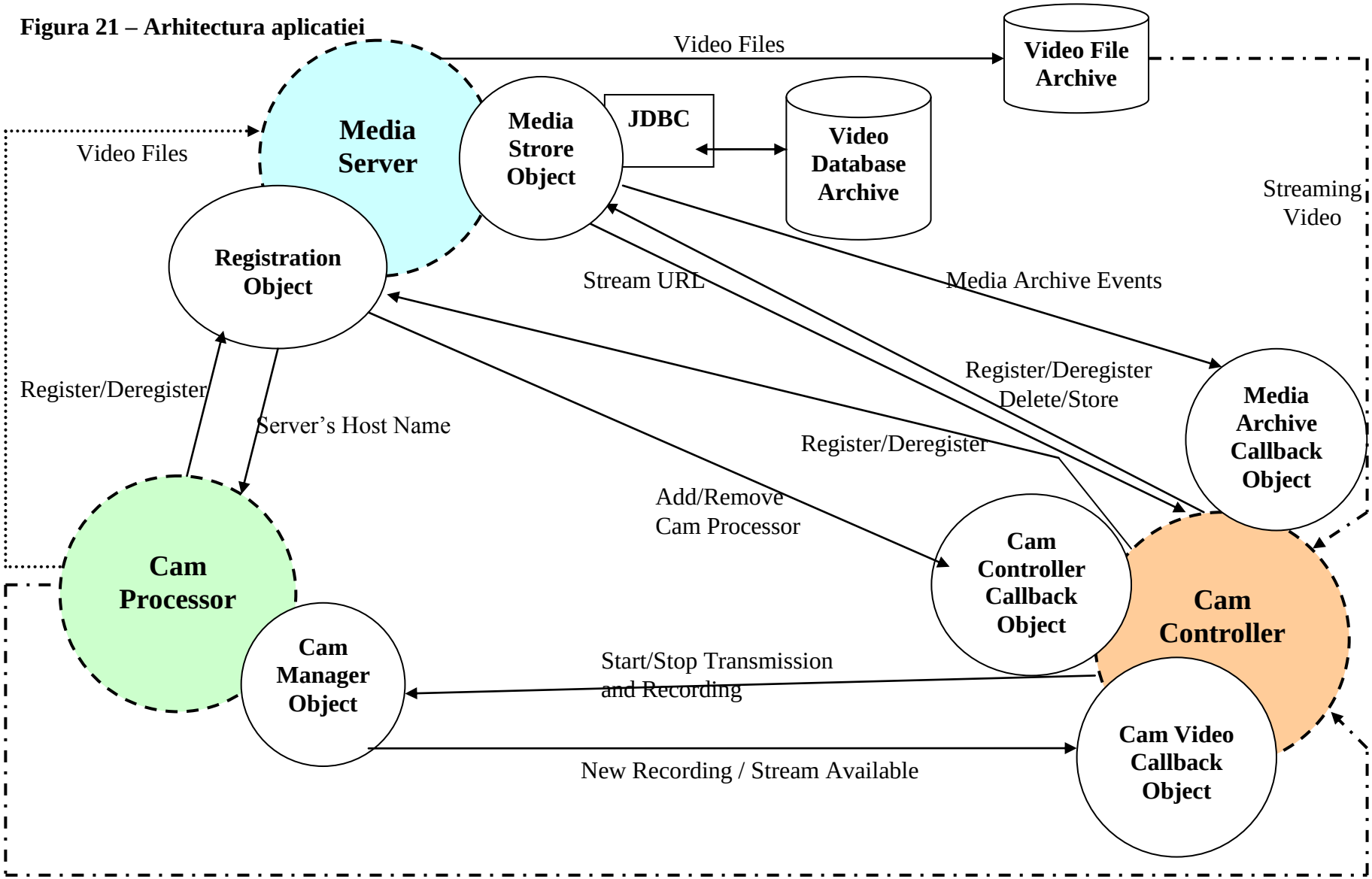

Figura de mai sus reprezintă arhitectura detaliată a sistemului S.V.L.W.. Se observă ca toate cele trei entităţi folosesc obiecte la distanţă pentru interacţiunea dintre ele. În figură sunt reprezentate în diferite forme legăturile între cele trei entităţi:

 modul aplicaţie

 obiecte CORBA – obiecte la distanţă

 conexiune socket

 apel de metoda

 transfer RTP

Pentru simplificarea schemei nu s-au reprezentat decât cate un modul de captura şi de control, chiar daca aplicaţia permite fără configurări suplimentare adăugarea de noi module, având constrângerea ca acestea să ruleze pe calculatoare gazdă diferite.

IV. Proiectare de detaliu

În acest capitol va fi descrisă etapa de implementare din procesul software ce stă la baza sistemului de supraveghere S.V.L.W.. În ordine vor fi descrise: structura programului (diagrame UML de clase şi de secvenţa), interfaţă cu utilizatorul, structuri de baze de date, comunicarea cu alte sisteme.

1. Structura programului

Aplicaţia în sine este împărţită pe mai multe module, după cum a fost descris în capitolele anterioare, fiecare dintre aceste module având posibilitatea de a funcţiona independent de celelalte, dar fără să mai poată satisface multe din funcţiile pentru care a fost proiectat. Fiecare modul la rândul său are mai multe submodule, reprezentate prin pachete. Fiecare pachet conţine clase de acelaşi gen, ele împreuna conferind sistemului o anume funcţionalitate.

1.1. Modulul server de date

Acest modul reprezintă modulul central al sistemului, practic orice informaţie ce se transmite în sistem se face prin obiectele acestui modul. Pachetul principal este pachetul *webS* organizat în mai multe subpachete, care grupează clasele Java după funcţionalitate. Un pachet generat automat pentru comunicarea obiectelor la distanţa este pachetul *idlPack*. Acesta este rezultatul interfeţelor scrise pentru obiectele la distanţă, generându-se prin compilarea fişierului de descriere a interfeţelor *“.idl”*. Compilarea va trebui făcută înaintea rulării proiectului în sine.

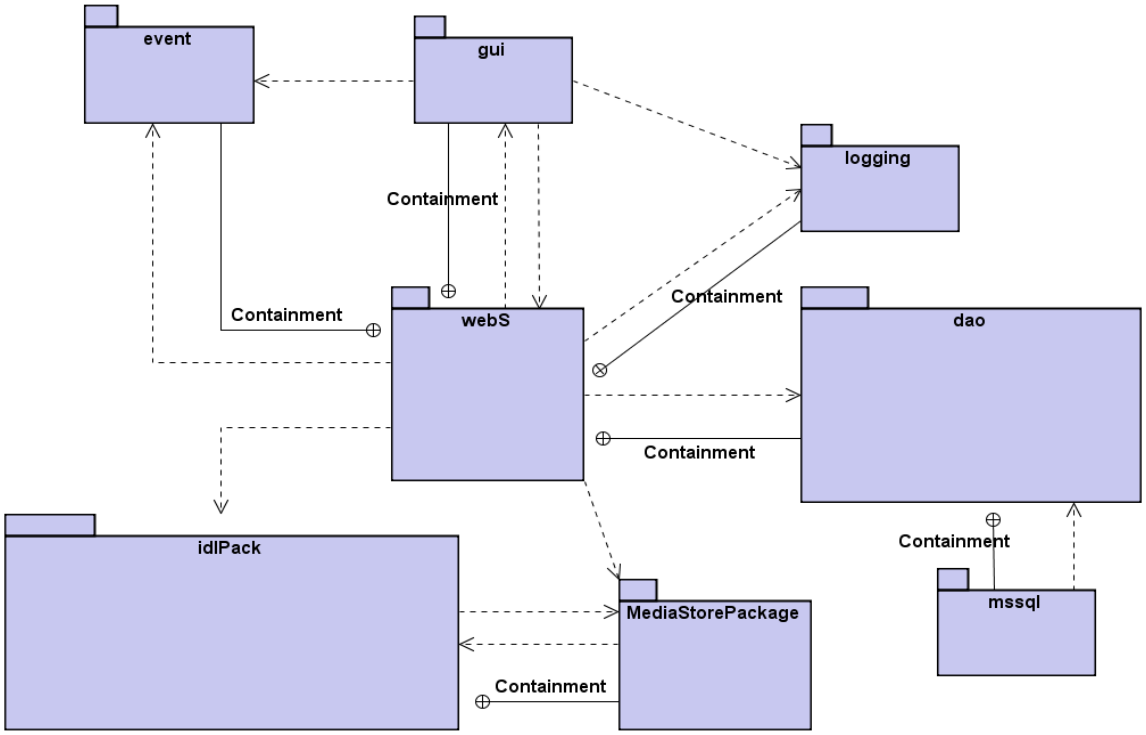

Figura 22 – Media Server-Diagrama de pachete

- ***Pachetul webS***

Reprezintă pachetul principal al modulului. Conţine o serie de subpachete şi în plus câteva clase cu caracter funcţional, ce pot fi clasate în nivelul controlerului în arhitectura MVC.

În diagrama UML sunt reprezentate doar legăturile dintre clasele din pachetul webS nu şi legăturile cu clasele din subpachete.

- *clasa Server*

Clasa main a modulului server de date, cu ajutorul ei pornindu-se aplicaţia. Conţine metode de iniţializare a serviciului de nume CORBA, de conectare la baza de date, de pornire a serverului de fişiere.

- *clasa MediaStoreImpl*

Este clasa implementată a interfeţei MediaStore din fişierul de declarări interfeţe. Implementează metodele de acces de la distanta la obiecte, pentru acţiunile asupra sursei de date.

- *clasa RegistrationImpl*

Implementarea interfeţei Registration. Are metode pentru înregistrarea de locaţii webcam şi de controlere, pentru ştergerea legăturii nume-referinţa, pentru returnarea şirului de webcam-uri ce capturează la un moment dat.

- *clasa FileServer*

Clasa destinata server-ului de fişiere. Porneşte un ServerSocket pe un anume port pentru a putea realiza transferul fişierelor înregistrate.

- *clasa VideoTransmiter*

Acoperă funcţia modulului de a transmite date de timp real către controler. Deschide o conexiune RTP şi transmite fişiere multimedia spre a fi revizionate.

- ***Subpachetul webS.event***

Pachetul conţine interfeţe şi clase ce gestionează evenimentele ce pot să aibă loc în sistem: apariţia/dispariţia unui controler, apariţia/dispariţia unui webcam, apariţia/ştergerea unui fişier înregistrat.

- ***Subpachetul webS.logging***

Sistemul îşi loghează întreaga activitate, acest lucru fiind posibil cu ajutorul claselor din acest pachet.

- *clasa Logger*

Este o clasă ce înfăşoară logger-ul din log4j.jar. O astfel de abordare este bună în cazul în care la un moment dat se decide folosirea unui alt logger, schimbările necesare vor fi minime.

- *clasa LoggerFactory*

Este clasa care întoarce o instanţă de logger, a unei clase interne. Pentru utilizarea logging-ului trebuie configurate anumite proprietăţi, într-un fişier de *logging.conf*.

- *clasa MyLogger*

Clasa MyLogger este internă clasei LoggerFactory. Are rolul configurării logger-ului pentru a putea fi folosit.

Astfel, când va fi necesar un logger se va apela metoda getLogger, a clasei abstracte LoggerFactory, care returnează instanţa la un obiect de tip MyLogger

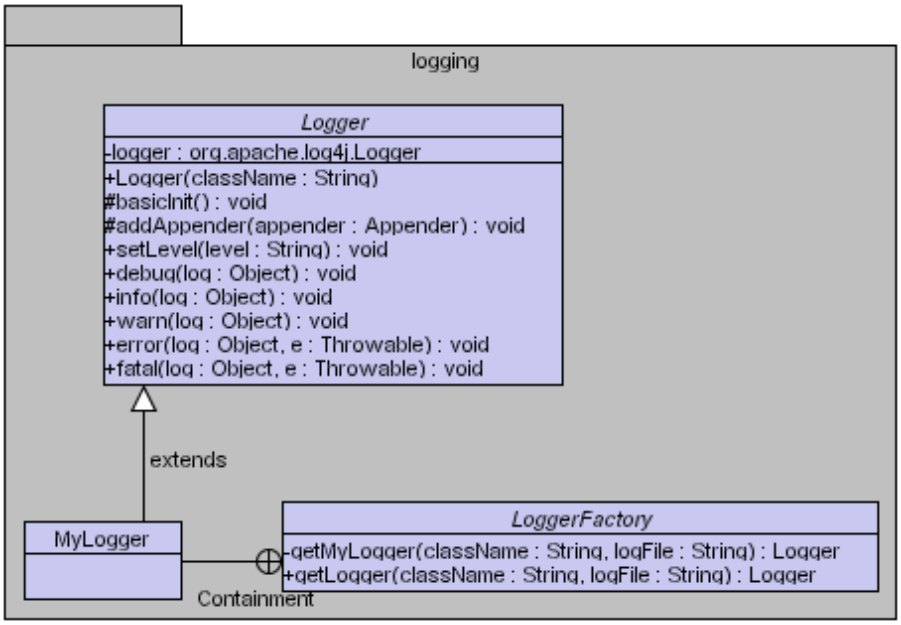

Figura 23 – Pachetul webS.logging

- ***Subpachetul webS.gui***

Pachetul conţine clasele care construiesc interfaţa cu utilizatorul, deci clase ce moştenesc clase din pachete *java.awt* sau *javax.swing.*

- ***Subpachetul webS.dao***

Conţine clasele ce se ocupă cu managementul conexiunii cu bază de date. Structura pachetului şi logica organizării şi proiectării claselor este conform şablonului DAO(Data Access Object), astfel încât orice schimbare sau necesitate viitoare va fi mai uşor de gestionat. Necesitatea adaptării aplicaţiei pentru un alt tip de bază de date va fi foarte uşor de rezolvat, prin adăugarea, în eventualitatea că nu este deja adăugat, a unui pachet de clase care să facă posibilă legătura cu noul tip de bază de date.

- ***webS.dao.mssql***

Acest subpachet este reprezentat de clasele speciale scrise pentru MSSQL. Folosind aceste clase se va face legătura şi comunicarea (adăugări, ştergeri, reactualizări, interogări) cu baza de date MSSQL.

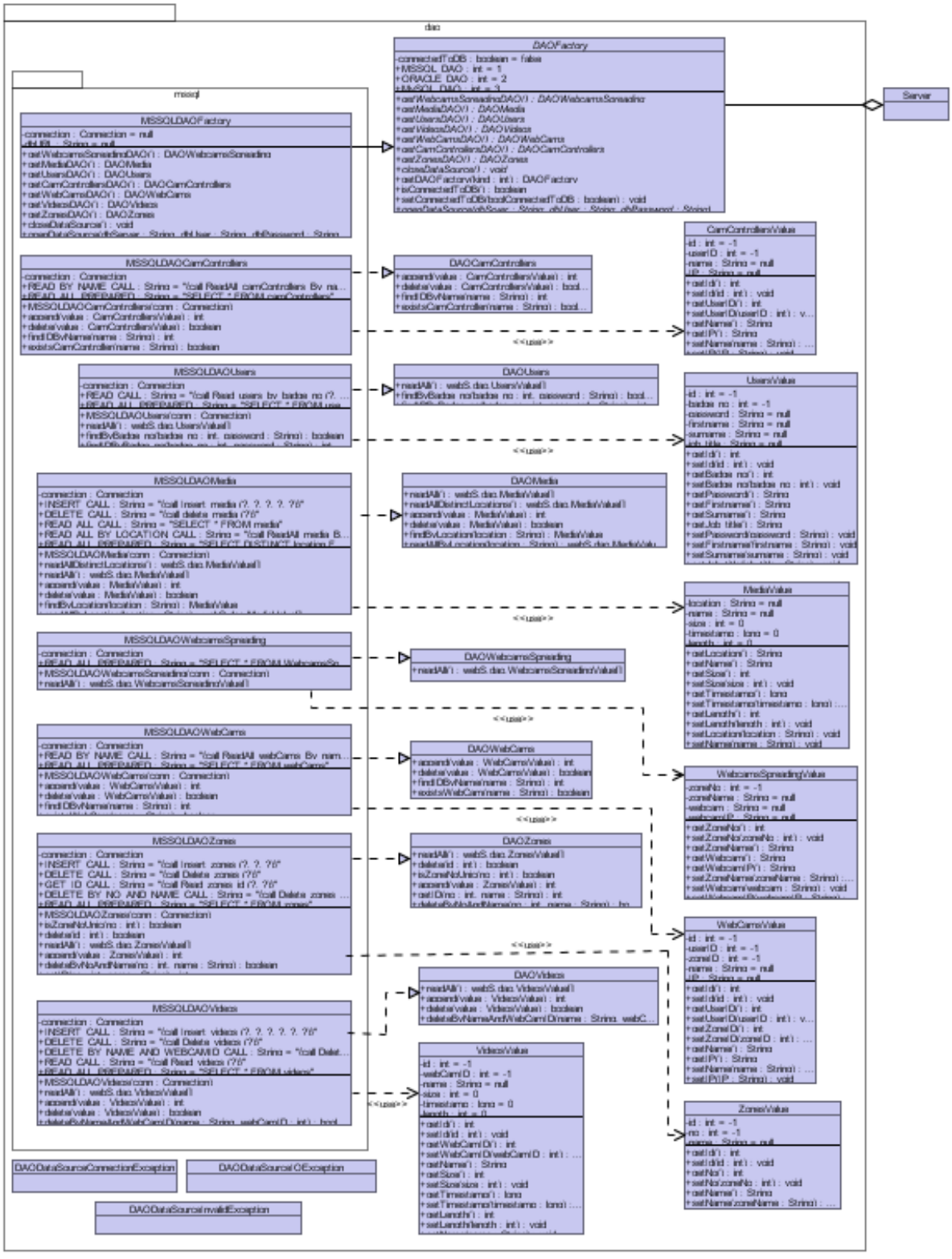

Figura 24 – Diagrama de clase a pachetului webS.dao

- ***Pachetul idlPack***
  - ***Subpachetul MediaStorePackage***

Aceste două pachete sunt cele generate automat pe baza fişierului de declarări a interfeţelor de obiecte la distanţă. Pe baza claselor din acest pachet se face comunicarea la distanţă.

1.2. Modulul de captura

Modulul de captură are ca prim rol captura de imagini de timp real de la dispozitivul de captură ataşat la calculatorul gazdă. Pe lângă acesta modulul trebuie sa fie capabil să trimită datele capturate spre a fi analizate de către modulul de supraveghere, să înregistreze pe discul local un fişier de date multimedia capturate iar apoi să transmită fişierul spre a fi stocat în arhiva de pe serverul de date.

Diagrama de pachete a acestui modul este descrisă mai jos şi se poate observa că include, la fel ca şi modulul anterior, pachetul idlPack, al claselor ce dau obiectele la distanţă. De asemenea subpachetul pachetul de *logging* care este practic acelaşi, logarea făcându-se pe acelaşi principiu, cu aceleaşi clase şi în acelaşi fel. Nivelul de logare se va seta separat pentru fiecare modul.

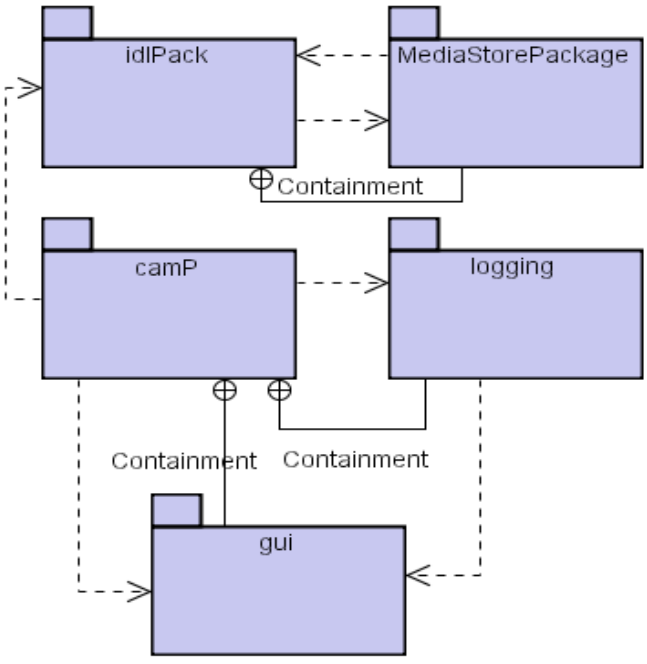

Figura 25 – CamProcessor-Diagrama de pachete

- ***Pachetul camP***

Este pachetul principal al modulului, conţinând clasa main, clase de control a funcţionalităţii, dar şi implementările obiectelor la distanţa specifice acestui modul.

În diagrama UML sunt reprezentate legaturile dintre aceste clase.

- *clasa CamProcessor*

Clasa main a modulului de captură, cu ajutorul ei pornindu-se aplicaţia. Conţine metode de iniţializare CORBA, de activare a obiectului la distanţă CamManager, de pornire a transferului de date de timp real, de pornire a serviciului de logare.

- *clasa ViewingThread*

Este clasa conţinută în clasa CamProcesor şi are rolul de a oferi posibilitatea utilizatorului de a vedea ceea ce se capturează.

- *clasa TransmitterThread*

Se ocupă de transferul efectiv de date multimedia către calculatorul gazdă al modulului controler.

- *clasa VideoTransmitter*

Pregăteşte transmiterea spre controler a datelor multimedia de timp real.

- *clasa RTPExport*

Clasa care se ocupă de înregistrarea datelor într-un anume interval de timp, pentru a putea fi mai apoi transmise spre server-ul de date.

- *clasa CamManagerImpl*

Implementarea obiectului la distanţă CamManager. Pe lângă implementarea metodelor obiectului la distanţă, mai oferă şi funcţionalitatea transferului fişierelor înregistrate.

- ***Subpachetul camP.logging***

Acelaşi ca cel de la modulul server de date.

- ***Subpachetul camP.gui***

Ca şi în cazul modulului server de date, pachetul gui conţine clasele ce formează interfaţa grafică a modulului de captură.

- ***Pachetul idlPack***
  - ***Subpachetul MediaStorePackage***

Aceleaşi cu cele de la modulul server de date.

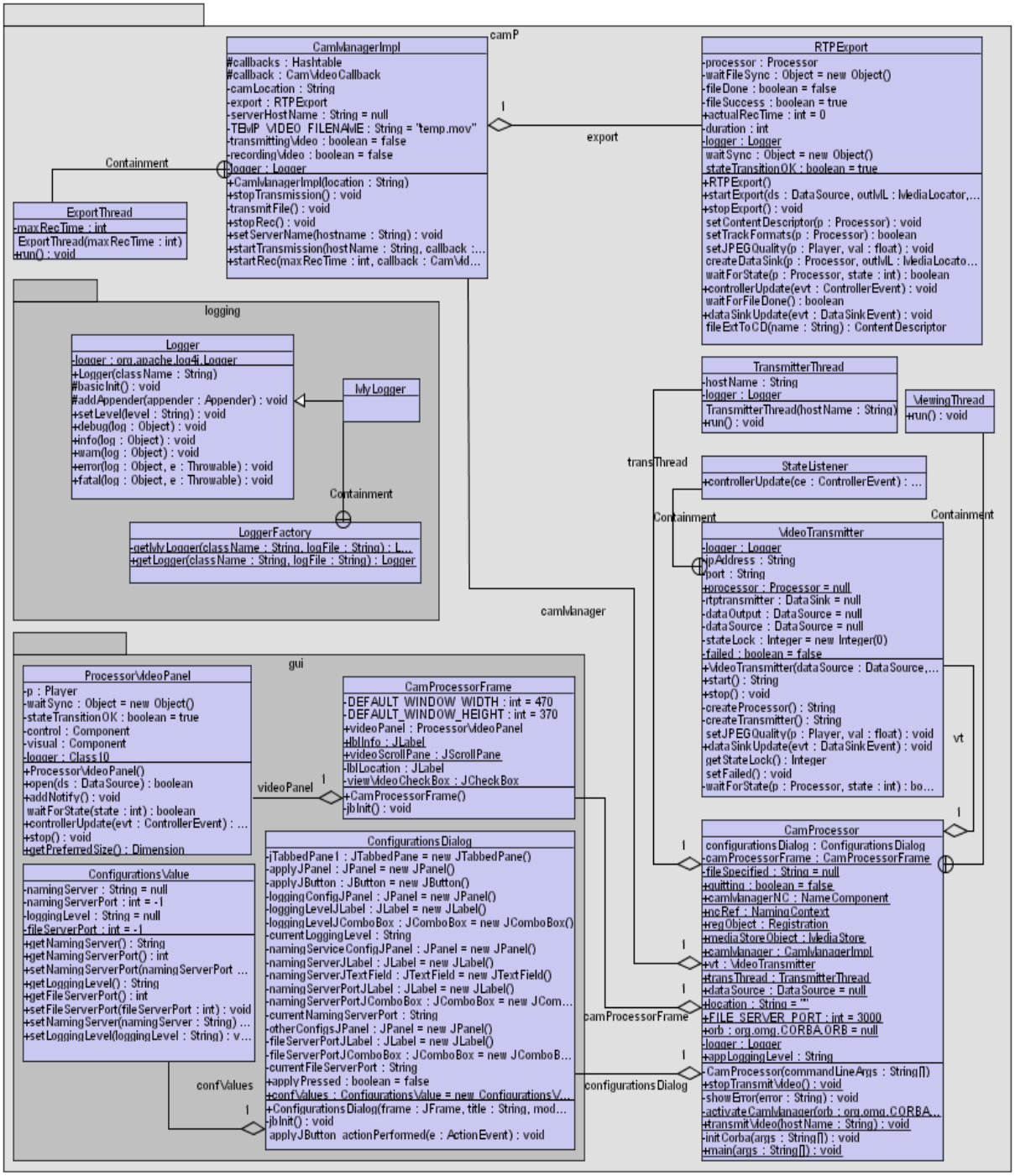

Figura 26 – Diagrama de clase a pachetului camP

Pentru a nu încărca foarte tare diagrama nu am mai reprezentat legăturile clasei *Logger*, din subpachetul *camP.logging*, deşi acestea există, obiecte logger fiind folosite în aproape toate clasele modulului. În plus mai există dependenţe cu clasele din pachetul *idlPack*, nici acestea nu sunt reprezentate pe diagrama de mai sus.

1.3. Modulul de control

Modulul de control oferă utilizatorului rolul de decizie în privinţa supravegherii. Oferă posibilitatea de a supraveghea diferite zone de activitate, posibilitatea deciziei înregistrării de imagini sau vizionării de fişiere deja înregistrate. În plus utilizatorul poate deschide un fişier video de pe discul local.

La fel ca la modulele anterioare, codul sursă este structurat pe pachete care conţin clase ce satisfac aceeaşi funcţionalitate în cadrul modului.

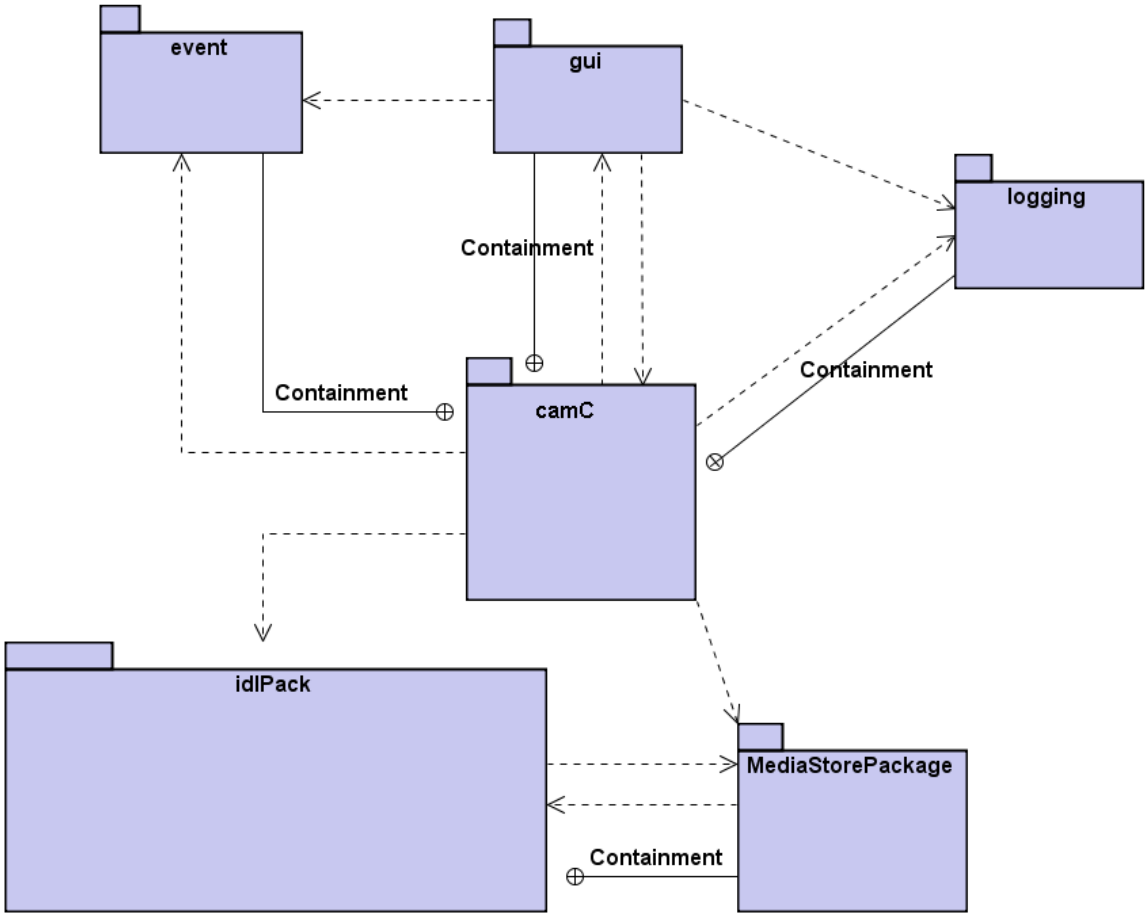

Figura 27 – CamController-Diagrama de pachete

- ***Pachetul camC***

Este pachetul principal al modulului, conţinând clasa main, clase de control ale funcţionalităţii, dar şi implementările obiectelor la distanţa specifice acestui modul. Controlul pe care îl are acest modul, la nivel utilizator, asupra aplicaţiei este realizat prin tehnica evenimentelor. Acestea sunt realizate prin callback-uri, în cazul acesta, obiecte la distanţă implementate la nivelul modulului de control, în pachetul principal.

- *clasa CamController*

Reprezintă clasa principală a modulului. În această clasă este iniţializat ORB-ul pentru comunicarea cu obiecte la distanţa şi legătura cu serverul de nume. Se realizează înregistrarea obiectelor la distanţa a căror implementări sunt în cadrul modulului de control.

- *clasa CamVideoCallback*

Este implementarea obiectului callback folosit pentru a gestiona evenimente de notificare a modulului de control cu privire la apariţia unei noi înregistrări.

- *clasa CamControllerCallback*

Este implementarea unui obiect la distanţă care are rolul de a anunţa modulul de control de apariţia a noi module de captură.

- *clasa MediaArchiveCallback*

Conţine metode ce anunţă modulul de control ca un nou fişier a ajuns în arhiva de fişiere şi în baza de date.

- ***Subpachetul camP.logging***

Acelaşi ca cel de la modulul server de date.

- ***Subpachetul camP.gui***

Interfaţa utilizator este realizată cu ajutorul claselor ce aparţin acestui pachet.

- ***Pachetul idlPack***
  - ***Subpachetul MediaStorePackage***

Aceleaşi cu cele de la modulele anterior prezentate.

Diagrama de secvenţă pentru cazul de utilizare *"ştergere fişier înregistrat"* este prezentată mai jos:

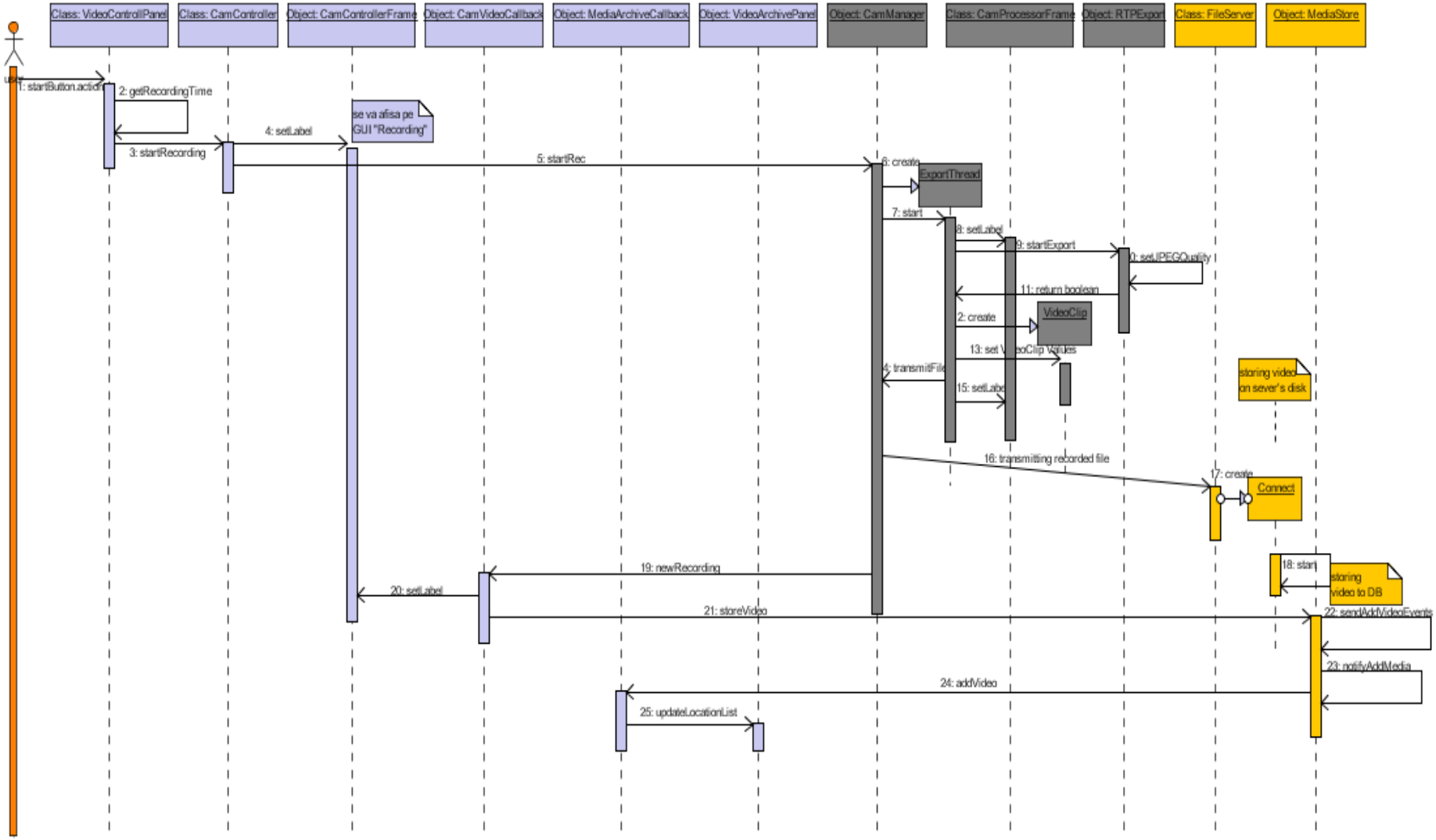

Figura 28 – Diagrama de secvenţa pentru inregistrare fişier

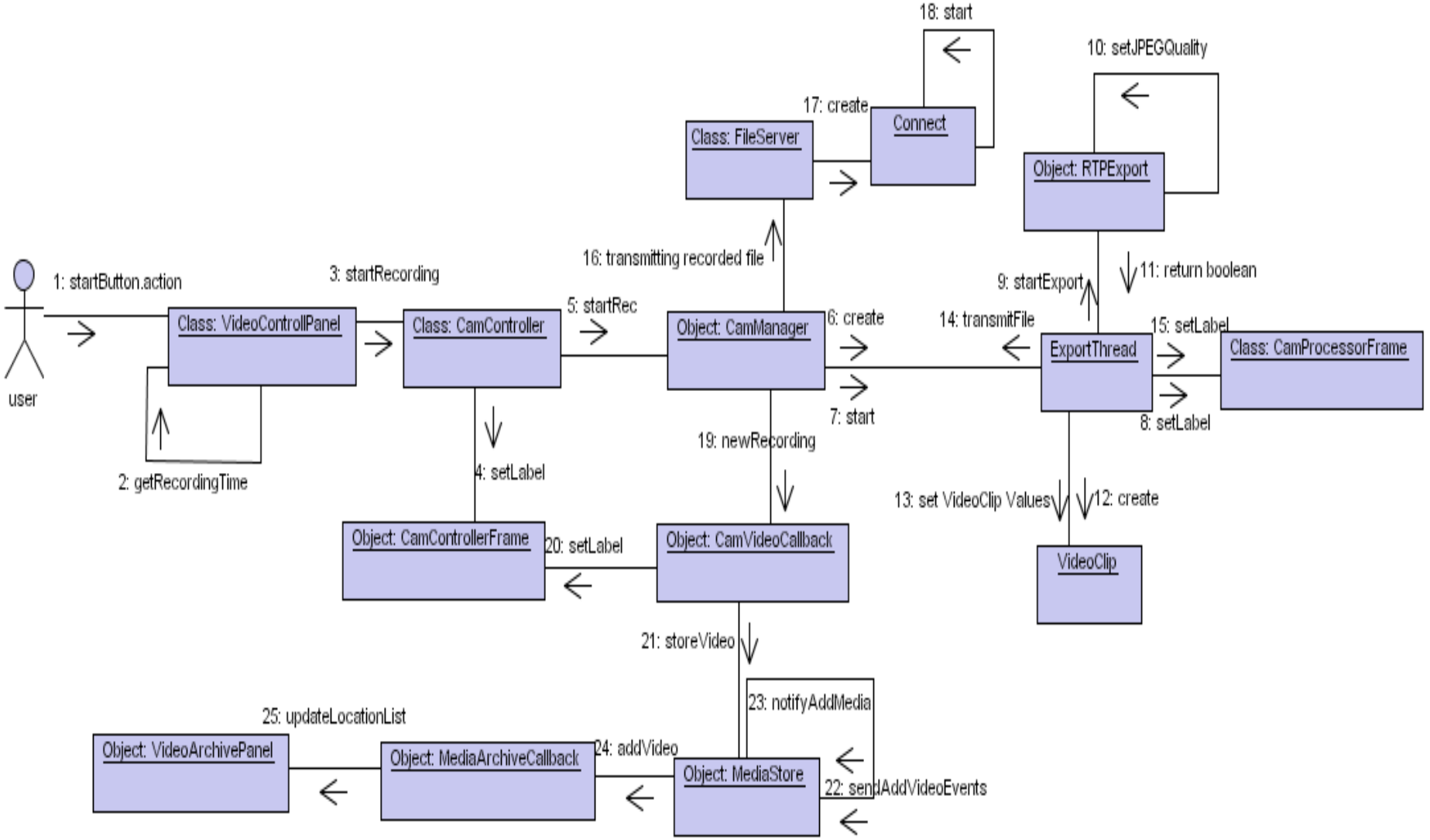

Figura 29 – Diagrama de colaborare pentru inregistrarea de fişier

2. Interfaţa cu utilizatorul

Interfaţa cu utilizatorul a fost realizată folosind clasele pachetului "*javax.swing*". S-a încercat respectarea regulilor de bază în ceea ce priveşte realizarea interfeţelor utilizator, fiind uşor de înţeles, fiecare buton având nume sugestiv pentru funcţia pe care urmează să o îndeplinească. Sistemul este conceput pe trei module fiecare având posibilitatea interacţiunii cu un eventual utilizator, astfel pentru fiecare modul a fost realizată o interfaţă separată.

2.1. Interfaţa modulului server de date

Ca la orice aplicaţie server-ul de date va avea interacţiune directă cu utilizatorul administrator. Administratorul de sistem trebuie să aibă posibilitatea schimbării unor configurări la pornirea aplicaţiei, astfel a fost realizată o fereastră de configurări prealabile pornirii efective a sistemului.

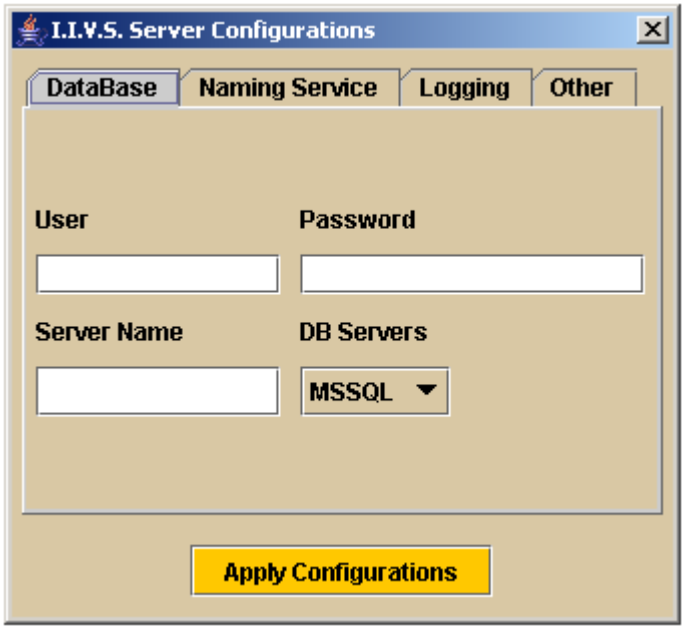

Figura 30 – Interfata configurari server de date

După cum se observă utilizatorul administrator are posibilităţi multiple de preconfigurare: configurarea legăturii cu baza de date, configurarea serviciului de nume (trebuie să cunoască numele şi portul pe care a fost pornit server-ul de nume), configurări privind modul de logare a activităţii şi alte configurări printre care setarea portului server-ului de fişiere.

Dacă datele introduse sunt valide atunci va apărea interfaţa proriu-zisă a server-ului de date. Se observă în mare măsură ce posibilităţi are administratorul.

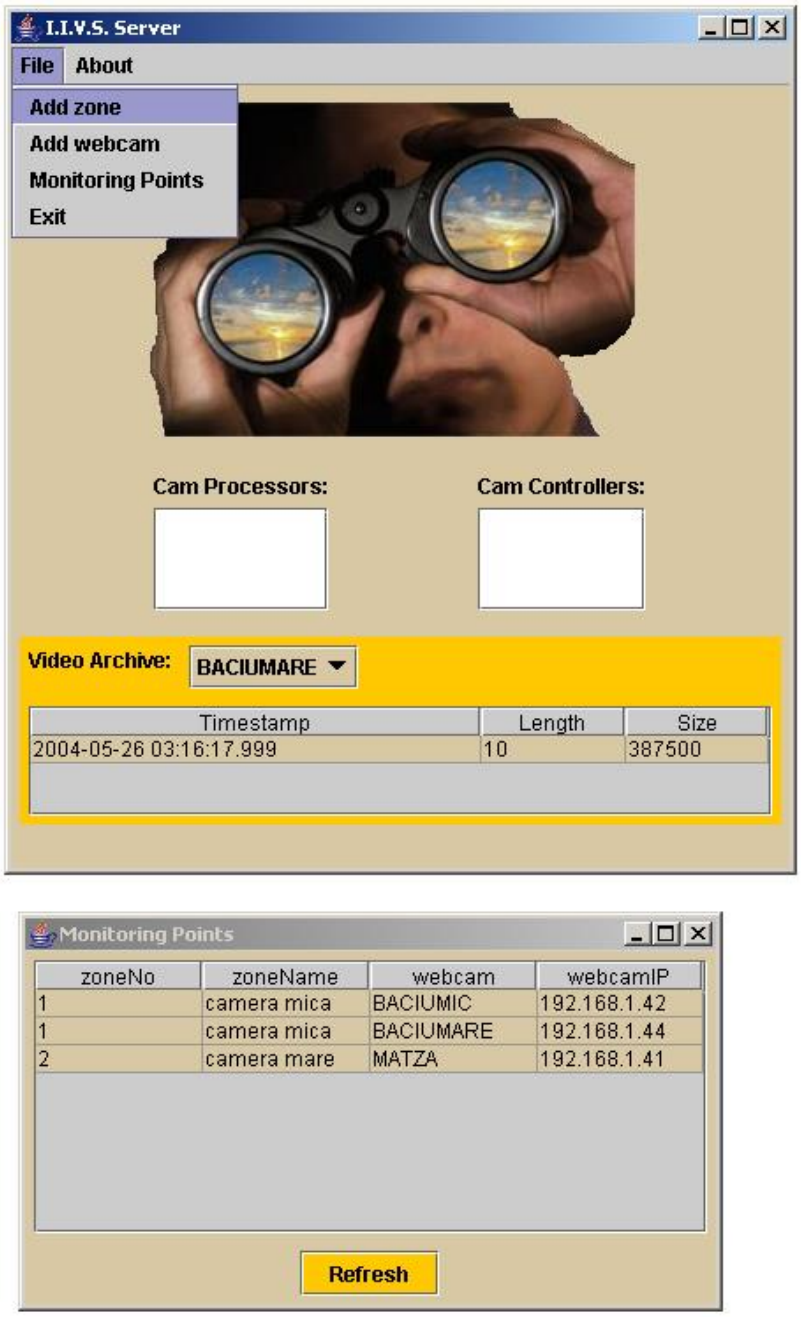

Figura 31 – Media Server-Interfata cu utilizatorul

Administratorul are o evidenţă a activităţii aplicaţiei, la un moment dat, prin analizarea interfeţei grafice, ce i s-a pus la dispoziţie.

2.2. Modulul de captura

Funcţiile ce trebuie să le îndeplinească modulul de captură sunt mai mult legate de buna funcţionare a sistemului, astfel nu este foarte dezvoltat pe planul interacţiunii cu utilizatorul. Acesta poate doar să privească ceea ce este capturat la un moment dat de dispozitivul de captură. Prealabil pornirii modulului, utilizatorul va trebui să cunoască unele configurări necesare funcţionării sistemului.

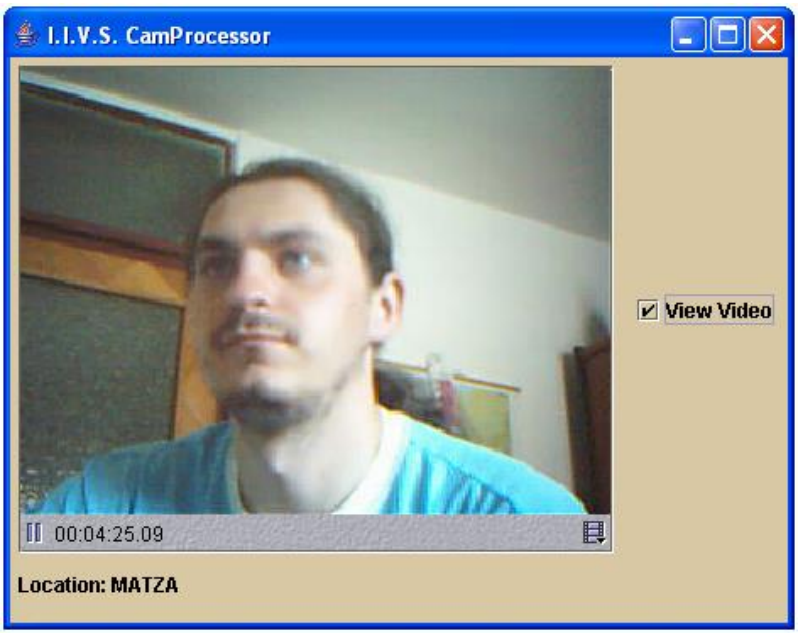

Figura 32 – CamProcessor-Interfata cu utilizatorul

Singura acţiune ce o poate avea utilizatorul, în cazul acestui modul este aceea de a viziona sau nu ceea ce se capturează de către dispozitivul de captură.

2.3. Modulul de supraveghere

Este modulul la care utilizatorul poate avea diverse acţiuni, între care cea mai importantă este aceea de supraveghere. Supraveghetorul are desfăşurata, pentru fiecare locaţie, listă fişierelor ce au fost înregistrate şi poate să le vizualizeze pentru a le reanaliza.

În orice moment supraveghetorul poate lua decizia înregistrării activităţii într-un fişier video, prin simpla apăsare a butonului *"Record"* şi setarea timpului de înregistrare. Transparent pentru utilizator, sistemul va realiza fişierul şi îl va anunţa că a fost înregistrat în arhiva de fişiere de pe server-ul de date.

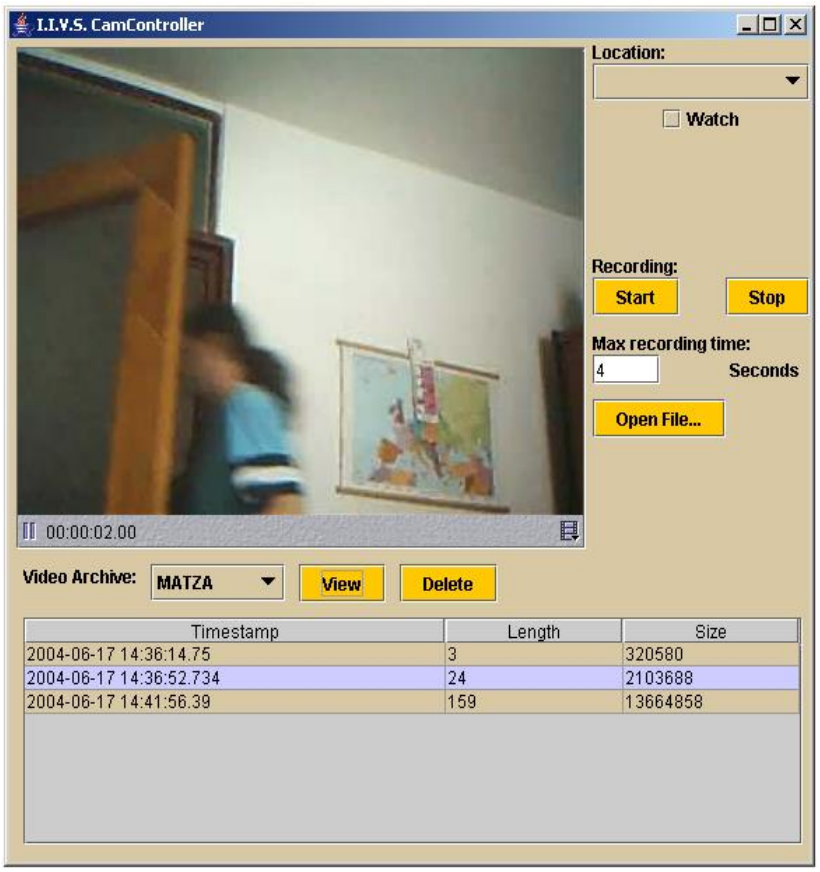

Figura 33 – CamController-Interfata cu utilizatorul

După cum se observă supraveghetorul are posibilitatea de a analiza fişierele din arhiva de fişiere în funcţie de momentul la care au fost înregistrate. Poate să vadă ce lungime are fiecare, deci poate să îşi dea seama între ce momente de timp a fost realizată înregistrarea activităţii de la locaţia respectivă.

3. Structuri de baze de date

Transparent pentru clienţii ce utilizează sistemul de supraveghere S.V.L.W. este conectarea la baza de date. Marea majoritate a sistemelor distribuite realizează o conexiune la o bază de date.

Pentru ca sistemul de faţă să funcţioneze în totalitate este nevoie să se realizeze mai întâi o bază de date cu cinci tabele. Fiecare tabel va conţine date despre diferiţi participanţi din sistem. Deoarece anumite informaţii sunt folosite foarte des s-au realizat vederi pentru a face mai uşoară interogarea bazei de date. De asemenea au fost realizate

proceduri stocate pentru obţinerea anumitor informaţii, această soluţie fiind preferată celei de a realiza interogările direct din codul sursă al aplicaţiei.

La pornirea server-ului de date administratorul va trebui să ştie un nume de utilizator şi o parolă, pentru accesul la baza de date a sistemului. Dacă aceste date nu sunt cunoscute nu se va putea realiza conexiunea. Astfel administratorul bazei de date trebuie să facă un pe *"login"* baza de date a sistemului şi apoi să îi comunice administratorului de sistem aceste date.

3.1. Diagrama bazei de date

Diagrama scoate în evidenţă legăturile între tabele, constrângerile existente în baza de date şi informaţiile pe care le ţine fiecare tabelă.

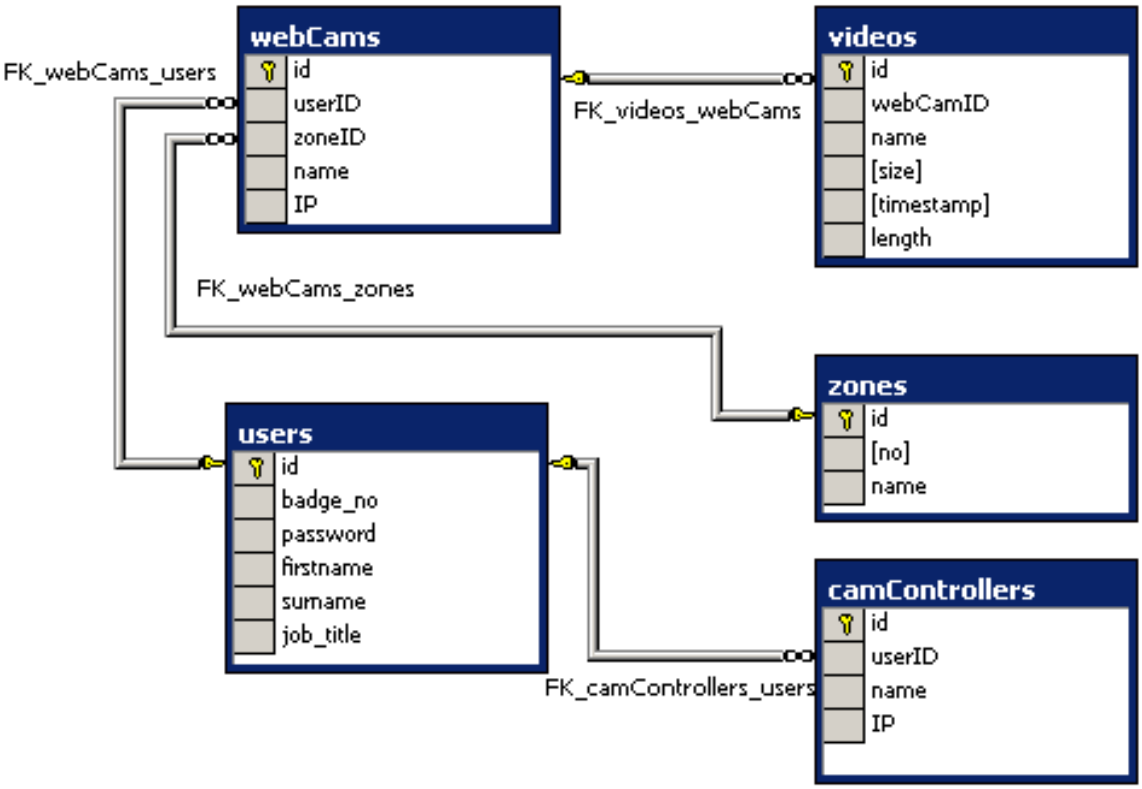

Figura 34 – Diagrama bazei de date

Pentru a oferi utilizatorilor posibilitatea de a şti cine, când şi unde a făcut o înregistrare, sau de a şti pe cine supraveghează, este necesar ca aceste informaţii să fie păstrate şi gestionate.

- ***tabela users***

Conţine informaţii despre utilizatorii care au drepturi de a face modificări în sistem. Se vor ţine şi câteva date suplimentare despre fiecare, pentru a putea verifica eventuale inconsistenţe de date fizice.

- ***tabela zones***

Orice spaţiu care trebuie supravegheat este împărţit în zone. Zonele pot să coincidă, chiar este indicat să coincidă, cu acelea de pe proiectul clădirii, sau schema de ieşire în caz de incendiu.

- ***tabela camControllers***

Conţine calculatoarele ce sunt luate în considerare ca fiind puncte de control. Instalarea unui modul de control este urmată de înregistrarea lui în baza de date.

- ***tabela webCams***

La fel ca şi tabela anterioară, tabela webCams conţine date despre punctele de supraveghere, având legătură cu tabela de zone. Astfel pentru fiecare punct de supraveghere se va cunoaşte şi zona pe care o supraveghează.

- ***tabela videos***

Este tabela cea mai importantă din diagramă, conţinând date despre fişierele video ce au fost înregistrate: numele fişierului, punctul de supraveghere de la care a fost înregistrat şi implicit zona de activitate, timpul la care s-a început înregistrarea, lungimea în secunde şi mărimea în bytes a fişierului.

3.2. Vederi

Vederile se folosesc în cazul în care sunt necesare aceleaşi informaţii de mai multe ori, în diferite cazuri. Sistemul de supraveghere S.V.L.W. foloseşte astfel de date. S-au construit doua astfel de vederi:

- ***media***

Este folosită frecvent pentru selectarea fişierelor unei locaţii de supraveghere, în cazul în care utilizatorul vrea să le revadă. Selecţia se face din trei tabele: *webCams*, *videos* şi *zones*.

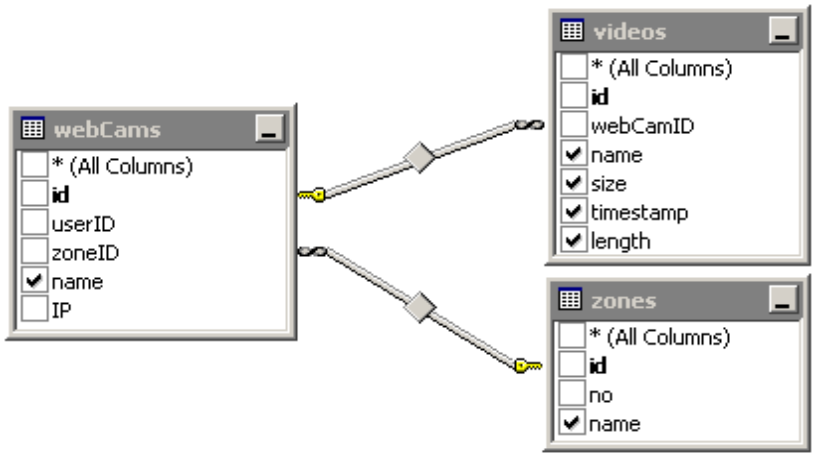

Figura 35 – Vederea media

- ***WebcamsSpreading***

Este folosită de către administratorul sistemului pentru a vedea ce puncte de supraveghere are în sistem.

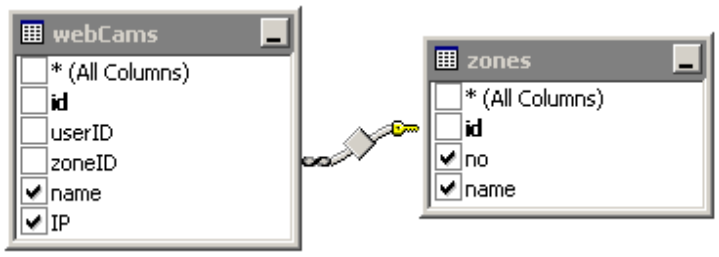

Figura 36 – Vederea WebcamsSpreading

Vederile reprezintă şi un nivel de securitate în plus, utilizatorul, fie el şi administrator de aplicaţie, nu va avea acces direct la datele din tabele ci doar la acelea ce i se furnizează prin intermediul vederii.

3.3. Proceduri stocate

Procedurile stocate reprezintă o alternativa utilă la interogarea bazelor de date făcute la nivelul codului sursă. Prin folosirea de proceduri stocate se va reduce timpul realizării interogărilor. Aplicaţia trimite severului de date parametrii şi numele procedurii stocate iar acesta întoarce rezultatul apelului de procedură.

În cazul sistemului S.V.L.W. au fost generate proceduri stocate pentru operaţiile necesare aplicaţiei:

- *ReadAll_media_By_location*
  - parametru de intrare: numele locaţiei webcam
  - răspuns (ieşire): fişierele înregistrate
- *ReadAll_webCams_By_Name*
  - parametru de intrare: numele locaţiei webcam
  - răspuns (ieşire): id-uri
- *Insert_videos*
  - parametri de intrare: datele necesare inserării
  - răspuns (ieşire): id-ul tuplei inserate
- *Delete_videos_By_name_And_webCamID*
  - parametru de intrare: numele fişierului şi locaţia camerei ce l-a înregistrat
  - răspuns (ieşire): -

Mai sunt şi altele folosite, dar acestea sunt cele mai relevante. Beneficiul adus de procedurile stocate este acela al separări clare a sarcinilor. Programatorul nu trebuie să ştie sintaxa SQL pentru a folosii apelul de proceduri stocate. În plus fiecare dintre componentele ce intră în contact în cadrul sistemului îşi face treaba cu uneltele proprii, fără să fie necesară cunoaşterea specifică a celuilalt.

V. Testare

Odată terminată faza de proiectare şi implementare sistemul este gata sa fie testat. Aceasta fază este foarte importantă în cadrul unui proces software, deoarece ar putea scoate în evidenţă anumite erori care să nu fi fost tratate în faza de implementare. De asemenea se pot descoperii anumite limitări ale sistemului, limitări ce ar putea fi date de diverşi factori: hardware sau software.

Prin testare se face o verificare a aplicaţiei pentru un set limitat de date. Doar în momentul în care setul de date de intrare este complet, acoperind toate cazurile de utilizare posibile, se poate spune ca sistemul funcţionează corect, dar cum acest lucru nu se poate realiza decât după o perioadă lungă de funcţionare se va demonstra corectitudinea dar fără garanţia absolută.

1. Tehnica de testare

În momentul testării pot fi urmăriţi mai mulţi factori ce sunt consideraţi critici pentru aplicaţia testată: timp de răspuns, memorie ocupată etc. Prin testare se pot evalua performanţele aplicaţiei. În plus testarea poate duce la descoperirea de erori care vor fi tratate înainte ca produsul sa fie livrat. Cu cât erorile sunt descoperite mai târziu cu atât costul tratării lor este mai mare.

Importanţa testării a dus la propunerea mai multor metode de realizare a acesteia. Testarea de jos în sus este metoda clasica de testare, ea constând în testarea modulelor şi apoi a sistemului în întregime. Testarea de sus în jos este posibila în cazul programării structurate, deci a conceperii descendente a sistemului. Ea se face odată cu realizarea modulelor, începând de la modulul principal apoi la modulele din primul nivel de subordonat şi aşa mai departe. Metoda de testare mixta presupune folosirea simultana a ambelor metode mai sus prezentate. Alegerea metodei de testare depinde de felul în care a fost concepută realizarea proiectului, o abordare ascendentă implică folosirea testării de jos în sus.

Testarea sistemului de supraveghere S.V.L.W. s-a făcut folosind ***testarea de jos în sus***, cu diverse dispozitive de captură, de la webcam-uri la camere video şi aparate foto ce permit conectarea la diferite porturi ale calculatorului şi suportă captura multimedia. Testarea a constat în supravegherea mai multor locaţii timp de câteva ore. S-au constatat probleme ce

erau deja cunoscute în domeniu, însa testarea a fost deosebit de utila deoarece am avut ocazia să observ comportarea sistemului in condiţii reale de funcţionare.

1.1. Aparatura folosită

S-au folosit pentru testarea sistemului trei tipuri de dispozitive de captură:

- ***Mustek PC Cam 300A***

Acest tip de dispozitiv de captură a fost cel mai folosit dar nu s-a dovedit a fi şi cel mai bun, deoarece calitatea imaginii nu este foarte bună şi scade exponenţial la lumina artificială.

- ***Logitech QuickCam Express Webcam***

Cele mai bune rezultate din punctul de vedere al calităţii imaginilor, în orice fel de condiţii au fost date de folosirea dispozitivului de mai sus. Acesta oferă, în timpul conectării sale la computer, posibilitatea de a fotografia.

- ***Fuji FinePix 2.0-Megapixel Digital Camera***

"Fuji FinePix 2.0-Megapixel Digital Camera" are ca prima facilitate partea de fotografiere, însă permite şi folosirea sa ca şi webcam. Calitatea oferita este una medie suferind în cazul existenţei mişcării.

- ***Sony DSC P32 Digital Camera***

Acest dispozitiv nu este unul din categoria celor descrise mai sus, pentru conectarea şi folosirea sa ca dispozitiv de captură de imagini fiind necesară instalarea de hardware suplimentar, respectiv o placa *"TV tunner"*. Calitatea imaginilor este bună, necesităţile sale hardware suplimentare nu au dus la pierderea calităţii imaginilor capturate.

1.2. Condiţiile de testare

Testarea s-a făcut în mai multe zile la rând într-un apartament cu doua camere, timp de câteva ore. S-au instalat dispozitive de captură pe două calculatoare din cele două camere, iar un al treilea era folosit pentru modulul de control, server-ul de date, server-ul de nume şi server-ul de baze de date, distribuirea sistemului fiind în acest caz restrânsă.

Cu ajutorul modulului de control au fost supravegheate cele două incinte, camere, realizându-se comutări de la o camera la alta şi înregistrări ale activităţii în perioade de timp

limitate. Deasemenea au fost revizualizate imaginile inregistrate pentru a observa daca au pierdut din calitatea din momentul capturii.

2. Probleme aparute

În timpul testării au fost sesizate o serie de probleme, în marea lor majoritate datorate unor mici neglijenţe la faza de implementare. Au apărut şi probleme ce nu ţin de dezvoltarea propriu-zisă a sistemului ci de uneltele folosite pentru a-l realiza. Astfel au fost descoperite limitări ale framework-ului JMF, în ceea ce priveşte compatibilitatea cu driver-ele unor dispozitive de captură din cele descrise mai sus.

Erorile majore au apărut la modulul de captură, care după o perioadă oarecare de rulare nu mai funcţiona rezultând un fişier de eroare ca cel prezentat mai jos, având numele ***"hs_err_pid2592.log"***:

An unexpected exception has been detected in native code outside the VM.
Unexpected Signal : EXCEPTION_ACCESS_VIOLATION (0xc0000005) occurred at PC=0xA782044
Function=[Unknown.]
Library=C:\WINDOWS\system32\GLZW.dll

NOTE: We are unable to locate the function name symbol for the error
just occurred. Please refer to release documentation for possible
reason and solutions.

Current Java thread:
at com.sun.media.codec.video.vcm.VCM.icLocate(Native Method)
at com.sun.media.codec.video.vcm.NativeDecoder.getSupportedOutputFormats(NativeDecoder.java:192)
at com.sun.media.GraphNode.getSupportedOutputs(GraphNode.java:89)
at com.sun.media.ProcessEngine$ProcGraphBuilder.findTarget(ProcessEngine.java:976)
at com.sun.media.SimpleGraphBuilder.doBuildGraph(SimpleGraphBuilder.java:220)
at com.sun.media.SimpleGraphBuilder.buildGraph(SimpleGraphBuilder.java:168)
at com.sun.media.ProcessEngine$ProcGraphBuilder.buildCustomGraph(ProcessEngine.java:1274)
at com.sun.media.ProcessEngine$ProcGraphBuilder.buildCustomGraph(ProcessEngine.java:1210)
at com.sun.media.ProcessEngine$ProcGraphBuilder.buildGraph(ProcessEngine.java:944)
at com.sun.media.ProcessEngine$ProcTControl.buildTrack(ProcessEngine.java:655)
at com.sun.media.PlaybackEngine.doRealize1(PlaybackEngine.java:326)
at com.sun.media.ProcessEngine.doRealize(ProcessEngine.java:77)
- locked <0x10de8a98> (a com.sun.media.ProcessEngine)
at com.sun.media.RealizeWorkThread.process(BasicController.java:1400)
at com.sun.media.StateTransitionWorkThread.run(BasicController.java:1339)

Dynamic libraries:
0x00400000 - 0x00406000 C:\WINDOWS\system32\java.exe
0x77F50000 - 0x77FF6000 C:\WINDOWS\System32\ntdll.dll

```
0x77E60000 - 0x77F45000     C:\WINDOWS\system32\kernel32.dll
0x77DD0000 - 0x77E5B000    C:\WINDOWS\system32\ADVAPI32.dll
……..

Heap at VM Abort:
Heap
 def new generation   total 1344K, used 0K [0x10010000, 0x10180000, 0x104f0000)
  eden space 1216K,   0% used [0x10010000, 0x10010000, 0x10140000)
  from space 128K,   0% used [0x10160000, 0x101603c0, 0x10180000)
  to   space 128K,   0% used [0x10140000, 0x10140000, 0x10160000)
 tenured generation   total 17032K, used 16868K [0x104f0000, 0x11592000, 0x14010000)
   the space 17032K,  99% used [0x104f0000, 0x11569258, 0x11569400, 0x11592000)
 compacting perm gen  total 7936K, used 7784K [0x14010000, 0x147d0000, 0x18010000)
   the space 7936K,  98% used [0x14010000, 0x147aa3d8, 0x147aa400, 0x147d0000)

Local Time = Mon May 24 17:09:12 2004
Elapsed Time = 113
#
# The exception above was detected in native code outside the VM
# Java VM: Java HotSpot(TM) Client VM (1.4.2_03-b02 mixed mode)#
---------------------------------------------------------------------------------------------------------------------------------------------------------------
```

Specificaţia JMF aminteşte de existenţa unor posibile incompatibilităţi, cu unele driver-e ale dispozitivelor de captură. Inconsistenţa framework-ului reprezintă o problemă deoarece un sistem de supraveghere trebuie sa ruleze fară opriri neaşteptate. Rezolvarea ar putea fi dată de folosirea unor dispozitive care prin folosiri repetate îşi dovedesc compatibilitatea cu Java Media Framework.

3. Rezultate experimentale

În urma testării sistemului s-au observat următoarele:

- încărcarea deosebită a staţiilor pe care rulează modulele sistemului duce la timpi de răspuns foarte mari
- necesitatea de spaţiu pe staţia gazdă a modulului de captură în cazul înregistrării de lungă durată (soluţie temporară: limitarea timpului de înregistrare la 10 minute)
- folosirea în internet are ca efect creşterea deosebită a întârzierilor
- traficul mare realizat la transmiterea în timp real (soluţie: reducerea numărului punctelor de control la zece)

- blocaje la urmărirea de către doua module de control a aceluiaşi modul de captură (soluţie: un modul de captură poate fi supravegheat de un singur modul de control)

Munca de testare şi corectare a erorilor a fost uşurată mult de folosirea in implementare a logării. Sistemul logându-şi activitatea oferă posibilitatea de a descoperi mult mai uşor cauza si locul erorilor. Chiar şi informaţiile ce nu au de a face cu erorile sunt la un moment dat utile.Exemplu de logare a activitaţii la modulul server de date:

```
INFO main webS.Server - Initializing...
INFO main webS.FileServer - File Server listening on port 3000 for incoming video files.
INFO main webS.gui.WebcamsSpreadingTable - Getting new data from participants view..
INFO main webS.MediaStoreImpl - Retrieve webcams spreading.
INFO main webS.gui.VideoArchivePanel - Retrieving archive locations from MediaStore...
INFO main webS.gui.VideoTable - Getting new data for location: BACIUMARE!
INFO main webS.gui.VideoArchivePanel - Selected archive location BACIUMARE
INFO main webS.gui.VideoTable - Getting new data for location: BACIUMARE!
INFO main webS.Server - Server is ready.
INFO AWT-EventQueue-0 webS.gui.VideoTable - Getting new data for location: MATZA!
INFO AWT-EventQueue-0 webS.gui.VideoArchivePanel - Selected archive location MATZA
INFO ORBacus:Server:ReceiverThread webS.RegistrationImpl - addCamController called.
INFO ORBacus:Server:ReceiverThread webS.MediaStoreImpl - addMediaArchiveListener called for baciumare
INFO ORBacus:Server:ReceiverThread webS.MediaStoreImpl - Free port: 1170
INFO ORBacus:Server:ReceiverThread webS.MediaStoreImpl - About to transmit file:
D:\Test\WebSServer\dist/video/BACIUMARE_Thu_Jun_17_14_19_03_EEST_2004.mov
INFO Thread-3 webS.MediaStoreImpl - About to start VideoTransmitter...
INFO Thread-3 webS.VideoTransmitter - Video transmitted as: JPEG/RTP, 320x240, FrameRate=2.7
INFO Thread-3 webS.VideoTransmitter - - Setting quality to 0.5 on
com.sun.media.codec.video.jpeg.NativeEncoder$1$QCA@186f3b3
INFO Thread-3 webS.VideoTransmitter - Transmitting video to rtp://baciumare:1170/video
INFO Thread-3 webS.MediaStoreImpl - Started VideoTransmitter.
INFO Thread-16 webS.MediaStoreImpl - getDuration  = 11009000000
INFO Thread-16 webS.MediaStoreImpl - getMediaTime = 0
INFO Thread-15 webS.MediaStoreImpl - getMediaTime = 459000000
INFO Thread-16 webS.MediaStoreImpl - getMediaTime = 808000000
INFO Thread-15 webS.MediaStoreImpl - getMediaTime = 1158000000
INFO Thread-16 webS.MediaStoreImpl - getMediaTime = 1895000000
INFO Thread-3 webS.MediaStoreImpl - About to stop vt...
INFO Thread-3 webS.VideoTransmitter - Stopping processor...
INFO Thread-3 webS.VideoTransmitter - Stopping rtptransmitter...
INFO Thread-3 webS.VideoTransmitter - Everything is stopped.
INFO Thread-3 webS.MediaStoreImpl - Stopped transThread
INFO ORBacus:Server:ReceiverThread webS.RegistrationImpl - removeCamController called.
INFO ORBacus:Server:ReceiverThread webS.RegistrationImpl - removeCamController called.
```

VI. Utilizare sistem

1. Instalare

1.1. Cerinţe hardware

Pentru utilizarea sistemului sunt necesare următoarele:

- câte un ***computer*** pentru fiecare locaţie ce urmează a fi supravegheată
- câte un ***webcam*** pentru fiecare staţie pe care va fi instalat modulul de captură
- câte un ***computer*** pentru fiecare punct de control
- un ***computer*** pentru server-ul de date (acesta poate fi instalat pe oricare dintre celelalte computer-e folosite în sistem)
- un ***computer*** pentru server-ul de baze de date
- un ***computer*** pentru server-ul de nume CORBA

1.2. Cerinte software

Pentru rularea corectă a aplicaţiei este mai întâi nevoie să se instaleze alte aplicaţii software pe care aceasta le foloseşte.

Mai întâi pentru a oferi suportul comunicării cu obiecte la distanţa va trebui instalat ***"ORBacus 4.0."***, sau o versiune mai nouă. Acesta poate fi luat de la adresa de internet *http://www.orbacus.com/*, fără a fi plătite nici un fel de taxe, produsul este gratis. În pachetul procurat vor exista instrucţiuni clare de instalare pe orice sistem de operare, tot pe site-ul amintit se oferă suport în cazul unor eventuale probleme la instalare.

O altă cerinţă software a sistemului S.V.L.W. ar fi aceea a instalării JMF, necesar pentru folosirea datelor multimedia de timp real. Pachetul JMF oferă suport pentru captură, procesare, vizualizare şi transfer multimedia şi poate fi procurat de la adresa *http://java.sun.com/products/java-media/jmf/2.1.1/download.html*, fără taxe suplimentare, în versiunea *2.1.1.e*. La fel ca şi produsul anterior şi pachetul JMF vine cu specificaţii clare de instalare pe mai multe sisteme de operare, în plus este cunoscut suportul pe care îl oferă Sun Microsystems.

Pentru o cât mai uşoară instalare a sistemului s-a folosit o unealtă de construcţie pusă la dispoziţie de *apache. "Ant 1.6.1."* poate fi procurat de la adresa http://ant.apache.org/, fără taxe suplimentare şi având specificaţii clare de instalare şi utilizare. În plus pe acelaşi site sunt puse la dispoziţie specificaţii extinse de folosire.

Se consideră implicită existenţa unui pachet *"j2sdk"* pe computer-ul gazdă al aplicaţiilor. În implementarea sistemului s-a folosit *"j2sdk1.4.2_03"*.

Odată rezolvate toate cerinţele harware şi software poate fi utilizat sistemul de supraveghere.[17]

2. Utilizare

Pentru utilizarea aplicaţiei clientul trebuie să copieze sursele dau fişierele bitecode într-un director creat de el, sau să dezarhiveze o eventuala arhiva. Pentru a porni aplicaţia utilizatorul trebuie doar să scrie *ant* la un terminal în directorul creat chiar de el.

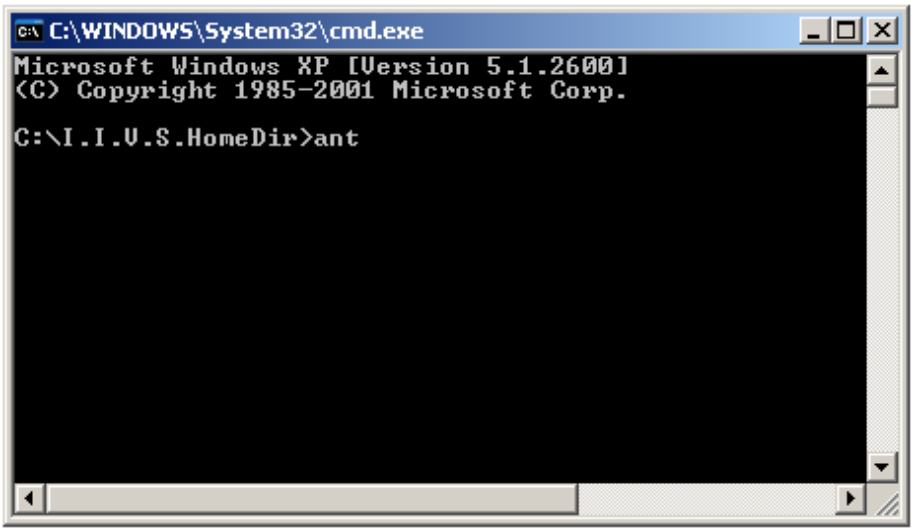

Figura 37 – Constructie S.V.L.W.

De aici urmează utilizarea propriu-zisă a aplicaţiei, moment în care utilizatorul este ajutat de interfaţa grafică ce i se pune la dispoziţie.

VII. Concluzii

Ideea de a încerca realizarea unui sistem ce transmite imagini de timp real între doua calculatoare, de la început, s-a transformat incet incet in realizarea unui sistem ce sa aiba aplicabilitate si sa satisfaca nevoi clare ale unei societati in plina dezvoltare.

Realizarea acestui proiect mi-a oferit sansa de a pune in practica teoria acumulata de-alungul anilor de studiu, dar si posibilitatea de a studia si utiliza cele mai noi tehnologii pentru dezvoltarea de sisteme distribuite multimedia.

În lucrarea de faţă s-a abordat domeniul transferului datelor multimedia de timp real, s-a proiectat un sistem de supraveghere ce atinge următoarele obiective şi cerinţe funcţionale:

- s-au implementat funcţiile de bază ale unui sistem de supraveghere (vizualizarea de imagini capturate, de la distanţă)
- există posibilitatea înregistrării şi stocării fişierelor de date multimedia
- s-a implementat posibilitatea revizualizării activităţii între anumite momente de timp, dacă a fost înregistrată
- s-a realizat un modul de vizualizare a imaginilor capturate la nivel local
- a fost conceput şi realizat un mecanism de autentificare a utilizatorilor ce schimbă configuraţia sistemului
- a fost implementată posibilitatea reconfigurării distribuirii resurselor ce le foloseşte sistemul, la nivelul evidenţei locaţiei lor
- sistemul îşi loghează activitatea în fişiere *"activity.log",* ceea ce face mai uşoară o eventuala necesitate de a descoperi o eroare.

Pe lângă atingerea acestor obiective funcţionale, ce sunt primordiale pentru orice sistem, au fost atinse şi câteva obiective nefuncţionale, datorită tehnologiilor folosite în implementarea sistemului. Folosirea CORBA aduce portabilitate absolută sistemului şi posibilităţi nelimitate de dezvoltare.

Performanţele obţinute sunt comparabile cu cele ale altor sisteme de acest fel. În dezvoltarea sistemului au fost descoperite unele erori pe care şi alţi dezvoltatori de aplicaţii multimedia le-au descoperit şi care sunt legate de utilizarea pachetului *Java Media Framework, versiunea 2.1.1.* Aceste probleme de stabilitate pe care le întâmpină JMF nu au fost încă rezolvate, dar problema este cunoscută şi probabil în următoarea versiune a JMF, Sun Microsystems va acoperii şi aceste neajunsuri.

Posibilitati de extindere ulterioara

Dătorită tehnologiilor folosite in implementare, adăugarea de noi module nu ridică nici un fel de probleme, dispare chiar si contrângerea folosirii aceluiaşi limbaj de programare sau a instalării pe acelasi sistem de operare.

Există mai multe direcţii in care sistemul de supraveghere S.V.L.W. ar putea fi dezvoltat. O scurtă descriere a acestora am să fac in cele ce urmează:

- realizarea unui mecanism de întrerupere a fişierului înregistrat la o anumită dimensiune şi reluarea înregistrarii într-un fişier nou (astfel s-ar elimina contrângerea cu privire la spaţiul necesar pe staţia gazdă a modulului de captură)
- prevederi la nivel de calitate a serviciului, ar duce la creşterea calităţii imaginilor de timp real transmise si la eliminarea contrângerii cu privire la conectarea unui singur modul controler la un modul de captură
- implementarea unui mecanism de detecţie a mişcării
- folosirea transferului multicast in cazul necesităţii amintite la punctul anterior
- multiplicarea server-elor de date, ar rezolva problema acceselor concurente multiple la arhiva de fişiere
- realizarea unui *„cluster”* de server-e de date media, ar extinde posibilităţile sistemului

Sisteme similare

Ideea realizării sistemelor de supraveghere nu este una noua, au fost încercate o serie de soluţii. S-a ajuns la realizarea de dispozitive hardware speciale pentru acest fel de activitate. Exista câteva sisteme de supraveghere care rezistă de ani buni pe piaţa. Marea majoritate realizează transmitere de imagini alb-negru prin reţea la un “framerate” foarte mic pentru a avea fluenţa imaginile de timp real si pentru a suporta transferul reţeaua de comunicaţie.

- **WebCCTV**
    - Un sistem de supraveghere ce rulează pe un dispozitiv dedicat (WebCCTV 4 Server)
    - poate avea conectate simultan 4 camere video prin interfaţa BNC
    - transmite informaţii in reţea pentru mai mulţi utilizatori
    - posibilitate de stocare pana la 40GB
- **VPON Camera Server**

- sistem de supraveghere pe baza de hardware dedicat
- permite conectarea de până la 6 camere video la calculatoare din aceeaşi reţea
- are posibilitatea transmiterii datelor stocate înspre alte locaţii

- **Web-based Video Surveillance System**
  - necesita instalarea unui server video
  - clienţii se pot conecta folosind orice browser
  - permite monitorizarea mai multor locaţii simultan
  - permite realizarea de *zoom in* si *zoom out* pe imaginile de timp real transmise

Toate aceste sisteme au un factor comun, dat de faptul ca folosesc unitaţi hardware dedicate, ce limitează şi numărul punctelor ce pot fi supravegheate.

VIII. Bibliografie

[1] Calin Marin Vaduva, ***Programarea in Java***, *editura Albastra, 2000*

[2] I. Salomie - ***Tehnici Orientate Obiect***, *editura Microinformatica, 1995*

[3] Bruce Eckel, ***Thinking in Java 2nd and 3rd edition***, *MindView, Inc*

[4] M. C. Chan, S. W. Griffin, A. F. Iasi, ***Java 1001 Secrete pentru programatori***

[5] Bruce Eckel, ***Thinking in Enterprise Java,*** *MindView, Inc*

[6] Robert Dollinger, ***Baze de date si gestiunea tranzactiilor***, *editura Albastra, 2000*

[7] Robert Dollinger, ***Utilizarea Sistemului SQL Server***, *editura Albastra, 2002*

[8] Eneia Todoran, ***Inginerie software***, *editura Mediamira, 2001*

[9] ***JMF API Guide,*** *Sun Microsystems, Inc, November 19, 1999*

[10] ***Orbacus 4.1.2. User Guide***, *IONA Technologies PLC, October 01, 2003*

[11] ***IDL to Java Language Mapings***, *June 1999*

[12] ***Naming Service Specification***, *September 2002*

[13] ***CORBA: Core Specification***, *December 2002, OMG*

[14] http://www.faqs.org/rfcs/rfc1889.html

[15] *http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html*

[16] *http://java.sun.com/blueprints/patterns/MVC-detailed.html*

[17] *http://ant.apache.org/manual/*

[18] *http://www.iso.org*

IX. Anexe

1. Fişierul de declarare a interfeţelor

webS.idl

```
/**
 * IDL interface definitions for WebS system.
 *
 * Copyright (C) 2004 by Ciprian Baciu
 * Technical University of Cluj-Napoca
 * May 2004
 *
 * Compile with Orbacus: jidl webS.idl
 */


module idlPack {
  typedef sequence<string> StringArray;  // Unbounded "array" of strings

  struct VideoClip {
    string location;   // Location of the video
    string name;  // Video name determined by timestamp and modified like so: Mon_Oct_12_02_55_13_CST_2002
    long   size;            // Size of video in bytes
    long   length;          // Length in seconds
    long timeStamp;//Difference in milliseconds between the current time and midnight on 1-1-1970UTC
  };

  //************************************************************************

  interface CamVideoCallback {
    // Notify CamController of a new video clip that has been recorded.
    oneway void newRecording(in VideoClip video);

    // Notify CamController of video that is being streamed to the given URL.
    oneway void videoAvailable(in string url);
  };

  //************************************************************************

  interface CamManager {
    // Start streaming video to given host name.  Callback's videoAvailable called when video
    // starts being transmitted.  Returns false if video can't be transmitted because CamProcessor
    // is busy transmitting elsewhere, true otherwise.
    boolean startTransmission(in string hostName, in CamVideoCallback callback);

    // Stop current transmission of video.
    oneway void stopTransmission();

    // Start recording video from web cam to file for a maximum of milliseconds.  The callback
    // is used to notify the CamController when the recording has been completed.
    boolean startRec(in long milliseconds, in CamVideoCallback callback);

    // Stop recording video.
    oneway void stopRec();
  };

  //************************************************************************

  interface MediaArchiveCallback {
    // Notify CamControllerProcessor that a video has been archived for this location
    oneway void addVideo(in string location);

    // Notify CamControllerProcessor that a video has been deleted at this location
    oneway void removeVideo(in string location);
  };
```

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

  interface MediaStore {
    exception LocationsUnavailable{};
    typedef sequence<VideoClip> Clips;

    // Register for listening for media archive events
    oneway void addMediaArchiveListener(in string hostName, in MediaArchiveCallback callback);

    // Deregister for media archive events
    oneway void removeMediaArchiveListener(in string hostName);

    // Return unbounded array of web cam names from the database. LocationsUnavailable if error //retrieving locations.
    StringArray getArchiveLocations() raises(LocationsUnavailable);

    // Return information from database about all archived media at a given location
    void getAllVideoDetails(in string location, out Clips videos);

    //Deletes a set of videos for a location from the database.  Return true if successful, false otherwise.
    boolean deleteVideos(in string location, in StringArray names);

    // Store a video in the database.
    oneway void storeVideo(in VideoClip videoRec);

    // Start streaming a particular video via RTP to the given destination IP address.
    string getVideoStream(in string location, in string name, in string destIPaddr);
  };

  //*************************************************************************

  interface CamControllerCallback {
    // Notifies all CamControllers that a new CamProcessor is available
    oneway void addWebCam(in string location);

    // Notifies all CamControllers that a CamProcessor is no longer available
    oneway void removeWebCam(in string location);
  };

  //*************************************************************************

  interface Registration {
    // Notify MediaServer that a new CamController has started
    oneway void addCamController(in string hostName, in CamControllerCallback callback);

    // Notify MediaServer that CamController is going down
    oneway void removeCamController(in string hostName);

    // Notify MediaServer that a new CamProcessor is available
    oneway void addWebCam(in string location);

    // Notify MediaServer that a CamProcessor is no longer available
    oneway void removeWebCam(in string location);

    // Used by CamProcessor to get MediaServer's host name
    string getServerHostName();

    // Return array containing locations of all active CamProcessors
    StringArray getLocations();
  };
};
```

2. Clase ce implementează interfeţele pentru obiecte la distanţă

MediaStoreImpl.java

```
/*******************************MediaStoreImpl.java*******************************/

package webS;

import javax.media.*;
import javax.media.protocol.*;

import java.io.*;
import java.util.*;
//import java.sql.*;
import java.net.ServerSocket;

import webS.event.MediaArchiveListener;
import webS.event.MediaArchiveEvent;

import webS.dao.*;
import webS.logging.Logger;
import webS.logging.LoggerFactory;

import idlPack.*;
import idlPack.MediaStorePackage.*;


/**
 *
 *   This file contains the implementation of the MediaStore methods for storing/accessing video
 * to/from a database using JDBC, actually a DaoFactory object.
 *
 *   Methods are made synchronized so that two competing requests don't access the resources at
 * the same time.  For example, accessing the        database at the same time causes JDBC exceptions.
 * <p>
 * @author Ciprian Baciu<br>
 * Technical University of Cluj-Napoca<br>
 * May 2004
 */

public class MediaStoreImpl extends MediaStorePOA {

  /**
   * Stores listeners for NewVideo events
   * NOTE: Events are not possible with CORBA unless using Event or Notification Services
   *       or callback methods.  Callbacks are implemented in WebS for sending events from
   *       remote objects.
   */
  private Set mediaArchiveListeners;

  /**
   * Stores remote listeners for media events
   * NOTE: Events are not possible with CORBA unless using Event or Notification Services
   *       or callback methods.  Callbacks are implemented in WebS for sending events from
   *       remote objects.
   */
  private Hashtable mediaListeners;


  /**
   * The logger for this class
   */
  private static Logger logger;


  /**
   * Creates a <code>MediaStoreImpl</code> object and assign default values to the fields
   */
  public MediaStoreImpl() {
```

```
    mediaArchiveListeners = new HashSet(1);  // Only Media Server should get events
    mediaListeners = new Hashtable();
    try {
      logger = LoggerFactory.getLogger(MediaStoreImpl.class.getName(), "activity.log");
      logger.setLevel(Server.appLoggingLevel);
    }
    catch (IOException e) {
      e.printStackTrace();
    }

  }

  /**
   * Adds a listener - Register a listener
   * @param listener The listener to add
   */
  public void addMediaArchiveListener(MediaArchiveListener listener) {
    mediaArchiveListeners.add(listener);
  }

  /**
   * Iterate through list of NewVideoListeners and generate a new event for each
   */
  private void sendAddVideoEvents(String location) {
    Iterator i = mediaArchiveListeners.iterator();
    while (i.hasNext()) {
      MediaArchiveListener listener = (MediaArchiveListener) i.next();
      listener.addVideo(new MediaArchiveEvent(location));
    }
  }

  /**
   * Get the active participants from DB, that means webcams and their
   * positions in the building.
   * @return the active participants
   */
  public WebcamsSpreadingValue[] getWebcamsSpreading() {
    WebcamsSpreadingValue[] webcamsSpreadingValues = null;
    try {
      webcamsSpreadingValues = Server.daoFactory.getWebcamsSpreadingDAO().readAll();
      logger.info("Retrieve webcams spreading.");
    }
    catch (DAODataSourceIOException e) {
      logger.error("Couldn't retrieve webcams spreading from DB. ", e);
      //e.printStackTrace();
    }
    return webcamsSpreadingValues;
  }

  public ZonesValue[] getZones() {
    ZonesValue[] zonesValues = null;
    try {
      zonesValues = Server.daoFactory.getZonesDAO().readAll();
    }
    catch (DAODataSourceIOException e) {
      //e.printStackTrace();
      logger.error("Couldn't retrieve zones from DB. ", e);
    }
    return zonesValues;
  }

  /**
   * Iterate through list of MediaArchiveListeners and generate a new event for each
   */
  private void sendRemoveVideoEvents(String location) {
    Iterator i = mediaArchiveListeners.iterator();
    while (i.hasNext()) {
      MediaArchiveListener listener = (MediaArchiveListener) i.next();
      listener.removeVideo(new MediaArchiveEvent(location));
    }
  }

  //|||||||||||||||| Implementation of remote methods|||||||||||||||//

  /**
   * Register for listening for media archive events
```

```
 * @param hostName the host name of the listener
 * @param callback
 */
public synchronized void addMediaArchiveListener(String hostName, MediaArchiveCallback callback) {
  //interface MediaStore - idl
  logger.info("addMediaArchiveListener called for " + hostName);

  mediaListeners.put(hostName, callback);
}

/**
 * Deregister for media archive events
 * @param hostName the host name of the listener to be removed
 */
public synchronized void removeMediaArchiveListener(String hostName) {
  //interface MediaStore - idl
  logger.info("removeMediaArchiveListener called for " + hostName);

  mediaListeners.remove(hostName);
}

/**
 * Notify all CamControllers of new media from this location
 * @param location
 */
private void notifyAddMedia(String location) {
  Enumeration enum = mediaListeners.elements();
  logger.info("Notifying all cam controllers of new media from " + location);
  while (enum.hasMoreElements()) {
    MediaArchiveCallback callback = (MediaArchiveCallback) enum.nextElement();
    callback.addVideo(location);
  }
}

/**
 * Notify all CamControllers of deleted media from this location
 * @param location The location from were a file was deleted
 */
private void notifyDeleteMedia(String location) {
  Enumeration enum = mediaListeners.elements();
  logger.info("Notifying all cam controllers to delete media from " + location);
  while (enum.hasMoreElements()) {
    MediaArchiveCallback callback = (MediaArchiveCallback) enum.nextElement();
    callback.removeVideo(location);
  }
}

/**
 * Returns an array of String with the names of locations
 * @return an array of String with the names of locations
 * @throws LocationsUnavailable
 */
public synchronized String[] getArchiveLocations() throws LocationsUnavailable {
  //interface MediaStore - idl
  String[] loc = {""};

  try {
    // Get unique list of locations
    MediaValue[] mediaValue = Server.daoFactory.getMediaDAO().readAllDistinctLocations();

    Vector v1 = new Vector(10);
    for(int i = 0; i < mediaValue.length; i++) {
      v1.addElement(mediaValue[i].getLocation());
    }

    // Convert vector to array
    loc = new String[v1.size()];
    v1.copyInto(loc);
  }
  catch (DAODataSourceIOException e) {
    logger.error("General error retrieving data from database!.", e);
    //e.printStackTrace();
    throw new LocationsUnavailable();
  }

  return loc;
```

```
 }

 /**
  * Get all the video details for the location <code>location</code>
  * @param location the location of the videos
  * @param videos The parameter will be filled with the details of  all the videos for the location
  */
 public void getAllVideoDetails(String location, ClipsHolder videos) {
   //interface MediaStore - idl
   // Put into videos all video data from database
   Vector clips = new Vector();

   try {
     MediaValue[] mediaValue = Server.daoFactory.getMediaDAO().readAllByLocation(location);

     for(int i = 0; i < mediaValue.length; i++) {
       VideoClip video = new VideoClip();
       video.location = location;
       video.name = mediaValue[i].getName();
       video.size = mediaValue[i].getSize();
       video.length = mediaValue[i].getLength();
       video.timeStamp = mediaValue[i].getTimestamp();

       clips.addElement(video);
     }
     // Puts clips vector into output parameter

     videos.value = new VideoClip[clips.size()];
     clips.copyInto(videos.value);
   }
   catch(DAODataSourceIOException daodsioe) {
     logger.error("General error retrieving data from database!!.", daodsioe);
   }
 }

 /**
  * Store video data in database and store video as a file so it can be more quickly
  * transported to listeners using getVideo
  * @param videoRec the video to be stored
  */
 public synchronized void storeVideo(VideoClip videoRec) {
   //interface MediaStore - idl

   int webCamID = -1;
   VideosValue videosValue = new VideosValue();

   logger.info("storeVideo received new video:");
   logger.info("name: " + videoRec.name);
   logger.info("location: " + videoRec.location);
   logger.info("size: " + videoRec.size);
   logger.info("length: " + videoRec.length);
   logger.info("timestamp: " + videoRec.timeStamp);

   // Change name of video file according to timestamp.
   // Example: BACIUCOMPUTER_Fri_May_21_05_03_44_EEST_2004.mov

   // May need to wait until Server socket thread is able to completely download
   String fileName = "video/" + videoRec.location + ".mov";
   File videoFile = new File(fileName);
   String newName = "video/" + videoRec.location + "_" + videoRec.name + ".mov";

   boolean rename = false;

   try {
     rename = videoFile.renameTo(new File(newName));
     int tries = 0;
     while (tries < 5 && !rename) {
       Thread.sleep(1000);
       tries++;
       rename = videoFile.renameTo(new File(newName));
     }
   }
   catch (InterruptedException ex) {
     logger.error("Error while trying to wait for file to download.", ex);
     //ex.printStackTrace();
   }
```

```
    if (!rename) {
      logger.debug("Error trying to rename file from " + videoFile + " to " + newName);
      return;  // Don't save to database
    }

    if (!DAOFactory.isConnectedToDB()) {
      logger.debug("Can't store media file... not connected to database.");
      return;
    }

    try {
      webCamID = Server.daoFactory.getWebCamsDAO().findIDByName(videoRec.location);
    }
    catch (DAODataSourceIOException e) {
      logger.error("Some exception concerning webCams table consistence.", e);
      //e.printStackTrace();
    }
    logger.info("webcamID:" + webCamID);
    videosValue.setWebCamID(webCamID);
    videosValue.setName(videoRec.name);
    videosValue.setSize(videoRec.size);
    videosValue.setTimestamp(videoRec.timeStamp);
    videosValue.setLength(videoRec.length);

    try {
      Server.daoFactory.getVideosDAO().append(videosValue);
    }
    catch (DAODataSourceIOException e) {
      logger.error("Some exception concerning inseration in videos table consistence.", e);
      //e.printStackTrace();
    }

    // Update MediaServer's archive video list
    sendAddVideoEvents(videoRec.location);

    // Notify all CamControllers of new media from this location
    notifyAddMedia(videoRec.location);

    logger.info("sendVideo is done.");
  }

  /**
   * Delete a set of videos from the specified location
   * @param location The location to delete from
   * @param names The set of video names
   * @return true if was a succesfull delete and false if not
   */
  public synchronized boolean deleteVideos(String location, String[] names) {
    int webCamID = -1;

    logger.info("deleteVideos with location: " + location);

    boolean success = true;

    try {
      webCamID = Server.daoFactory.getWebCamsDAO().findIDByName(location);
    }
    catch (DAODataSourceIOException e) {
      logger.error("Some exception concerning webCams table consistence.", e);
      //e.printStackTrace();
    }
    for (int i = 0; i < names.length && success; i++) {
      try {
        success = Server.daoFactory.getVideosDAO().deleteByNameAndWebCamID(names[i], webCamID);
      }
      catch (DAODataSourceIOException e) {
        logger.error("Some exception concerning inseration in videos table consistence.", e);
        //e.printStackTrace();
      }
      if (success) {
        // Delete file from video directory
        File f = new File("video/" + location + "_" + names[i] + ".mov");
        success = f.delete();
        if (!success) {
          logger.debug("Error deleting file: " + f.getAbsolutePath());
```

```
      }
    }
    else {
      logger.debug("Error trying to delete from database using delete command.");
    }
  }

  // Notify the MediaServer
  sendRemoveVideoEvents(location);

  // Notify all cam processors of deleted media from this location
  notifyDeleteMedia(location);

  return success;
}

/**
 * Start streaming a particular video via RTP to the given destination IP address.
 * @param location
 * @param name the name of the video that needs to be sent via RTP
 * @param destIPaddr The destination IP address
 * @return The RTP URL for accessing the streaming video that is sent to the
 *         given destination IP address.
 */
public synchronized String getVideoStream(String location, String name, String destIPaddr) {
  // Find a free port number on which to stream video
  int port = findOpenPort();

  String rtp = "rtp://" + destIPaddr + ":" + port + "/video";

  // Get current directory.  File is named like this: c:/../webSServer/video/location_name.mov
  String currentDir = System.getProperty("user.dir");

  String fileName = currentDir + "/video/" + location + "_" + name + ".mov";
  logger.info("About to transmit file: " + fileName);

  // Start tansmitting this video to this IP address
  TransmitterThread transThread = new TransmitterThread(destIPaddr, port, fileName);
  transThread.start();

  return rtp;
}

/**
 *   Finds a port number on the server which is not being used.  This would not
 * be necessary if only one CamController was running at a time on the same
 * computer as the server.  But since 2 or more CamControllers could be running
 * and accessing video at the same time, they would both need to access video
 * on different ports.
 *
 *  NOTE: Actually it's not the server's port that we need to find but a free
 * port on the CamController that we need to find.  Implement this later.
 *
 */
private int findOpenPort() {
  int port = 0;
  boolean freePort = false;

  while (!freePort) {
    try {
      // Test each port until we find one that is open for rtp transmission
      // Get free port from the system automatically
      ServerSocket s = new ServerSocket(0);
      port = s.getLocalPort();
      s.close();

      freePort = true;
    }
    catch (Exception e) {
      logger.error("", e);
      logger.info("Port " + port + " is taken.  Try next one...");
      // Try next one
      port += 2;
    }
  }
```

```
    logger.info("Free port: " + port);
    return port;
  }

  /**
   *
   * This inner class implements a thread for transmitting a file via rtp
   *
   */
  class TransmitterThread extends Thread {

    /**
     * destination IP address
     */
    private String ipAddr;

    /**
     * The port to use for transmision
     */
    private int port;

    /**
     * The filename to transfer
     */
    private String fileName;

    /**
     * Creates an <code>TransmitterThread</code> for a spesific address
     * @param ipAddr The destination address
     * @param port The port to use for transfer
     * @param fileName The filename to transmit
     */
    TransmitterThread(String ipAddr, int port, String fileName) {
      this.ipAddr = ipAddr;
      this.port = port;
      this.fileName = fileName;
    }

    /**
     * The run method
     * @see java.lang.Thread#run()
     */
    public void run() {
      logger.info("About to start VideoTransmitter...");

      DataSource ds;
      try {
        ds = Manager.createDataSource(new MediaLocator("file:/" + fileName));
      }
      catch (Exception e) {
        logger.error("Error creating data source from file: " + fileName, e);
        //e.printStackTrace();
        return;
      }

      VideoTransmitter vt = new VideoTransmitter(ds, ipAddr, Integer.toString(port));

      String result = vt.start();
      if (result != null) {
        logger.debug("Failed to start VideoTransmitter: " + result);
      }
      else {
        logger.info("Started VideoTransmitter.");
      }

      javax.media.Time duration = VideoTransmitter.processor.getDuration();
      if (duration == javax.media.Duration.DURATION_UNKNOWN) {
        logger.info("Unknown duration... run for 20 seconds.");
        duration = new javax.media.Time(javax.media.Time.ONE_SECOND * 20);
      }

      logger.info("getDuration  = " + duration.getNanoseconds());

      // Loop until we've exceeded the media's duration or media time is no longer inceremented.
      // This happens ofter when media time stops just short of duration time
      // NOTE: There's probobaly a better way to do this when RTPing a video file, but I haven't
```

```
      // been able to figure it out.

      long oldNano = -1;
      long nanos = VideoTransmitter.processor.getMediaNanoseconds();
      int numStucks = 0;

      while (nanos < duration.getNanoseconds() && oldNano != nanos && numStucks < 5) {
        logger.info("getMediaTime = " + nanos);
        oldNano = VideoTransmitter.processor.getMediaNanoseconds();

        try {
          //Thread.currentThread().sleep(1000);
          sleep(1000);
        }
        catch (InterruptedException ie) {
        }

        nanos = VideoTransmitter.processor.getMediaNanoseconds();

        if (nanos == 0) {
          // Taking a bit longer to get going
          nanos = 1;

          // Keep track of how many times we try this.  If we do it 5 times, there must
          // be something really wrong, and we should terminate the loop.
          numStucks++;
        }
      }

      // Must disconnect in order to release system lock on file.  Otherwise we won't be able to
      // delete the file later if the user wants the video deleted.
      ds.disconnect();
      ds = null;

      logger.info("About to stop vt...");
      vt.stop();            // Stop transmission thread  CAUSING PROBLEMS

      logger.info("Stopped transThread");
    }
  }  // end TransmitterThread class

}
```

RegistrationImpl.java

```
/*****************************RegistrationImpl.java*****************************/

package webS;

import java.util.*;
import java.io.IOException;

import webS.event.*;
import webS.event.WebCamListener;
import webS.logging.Logger;
import webS.logging.LoggerFactory;

import idlPack.*;

/**
 *  Contains the implementation of the remote Registration object which is
 * used by CamProcessors and CamControllers to register with the MediaServer.
 * Also contains event notification for MediaServer so it can update its
 * GUI when CamProcessors and CamControllers go up and down.
 * <p>
 * @author Ciprian Baciu<br>
 * Technical University of Cluj-Napoca<br>
 * May 2004
 */

public class RegistrationImpl extends RegistrationPOA {

  /**
```

```
 * CamControllers listening for new CamProcessors
 */
protected Hashtable camControllers;

/**
 * WebCams connected
 */
protected Vector webCamList;

/**
 * The Naming server host name
 */
protected String serverHostName = null;

/**
 * Listeners for WebCamEvents
 */
private Set webCamListeners;

/**
 * Listeners for CamControllerEvents
 */
private Set controllerListeners;

private Logger logger;

/**
 * Constructor
 */
public RegistrationImpl() {
  webCamListeners = new HashSet(1);  // Only Media Server should get events
  controllerListeners = new HashSet();
  camControllers = new Hashtable();
  webCamList = new Vector();
  try {
    logger = LoggerFactory.getLogger(RegistrationImpl.class.getName(), "activity.log");
    logger.setLevel(Server.appLoggingLevel);
  }
  catch (IOException e) {
    e.printStackTrace();
  }

}

/**
 * Adds a new listener to the set of WebCam listeners
 * @param listener A WebCam listener
 */
public void addWebCamListener(WebCamListener listener) {
  webCamListeners.add(listener);    // Register the listener
}

/**
 * Iterate through list of NewWebCamListeners and generate a new event for each
 * @param location The location of the new WebCam
 */
private void sendAddWebCamEvents(String location) {
  Iterator i = webCamListeners.iterator();
  while (i.hasNext()) {
    WebCamListener listener = (WebCamListener) i.next();
    listener.addWebCam(new WebCamEvent(location));
  }
}

/**
 * Iterate through list of WebCam listeners and generate a new event for each
 * @param location The location of the removed WebCam
 */
private void sendRemoveWebCamEvents(String location) {
  Iterator i = webCamListeners.iterator();
  while (i.hasNext()) {
    WebCamListener listener = (WebCamListener) i.next();
    listener.removeWebCam(new WebCamEvent(location));
  }
}
```

```
 /**
  * Register the listener
  * @param listener A listener
  */
 public void addCamControllerListener(CamControllerListener listener) {
   controllerListeners.add(listener);
 }

 /**
  * Iterate through list of NewCamControllers listeners and generate a new event for each
  * @param name The name of the CamController
  */
 private void sendAddControllerEvents(String name) {
   Iterator i = controllerListeners.iterator();
   while (i.hasNext()) {
     CamControllerListener listener = (CamControllerListener) i.next();
     listener.addCamController(new CamControllerEvent(name));
   }
 }

 /**
  * Iterate through list of CamControllers listeners and generate a new event for each
  * @param name The name of the CamController to be removed
  */
 private void sendRemoveControllerEvents(String name) {
   Iterator i = controllerListeners.iterator();
   while (i.hasNext()) {
     CamControllerListener listener = (CamControllerListener) i.next();
     listener.removeCamController(new webS.event.CamControllerEvent(name));
    }
 }

  //|||||||||||||||| Remote method implementations ||||||||||||||||//

 /**
  * Notify Server of cam controller so Server can notify it when web cams come up and down
  */
 public synchronized void addCamController(String hostName, CamControllerCallback callback) {
   logger.info("addCamController called.");

   camControllers.put(hostName, callback);

   sendAddControllerEvents(hostName);
 }

 /**
  * Tell Server that the CamController is going down so we don't continue to get notifications
  * @param hostName the host name of the CamController
  */
 public synchronized void removeCamController(String hostName) {
   logger.info("removeCamController called.");

   camControllers.remove(hostName);

   sendRemoveControllerEvents(hostName);
 }

 /**
  * Notify Server of new web cam so Cam Controllers know we're ready
  * @param location The location of the WebCam
  */
 public synchronized void addWebCam(String location) {
   // Add new web cam to list
   webCamList.add(location);

   Enumeration enum = camControllers.elements();
   logger.info("Notifying all cam controllers of new web cam: " + location);
   while (enum.hasMoreElements()) {
     CamControllerCallback callback = (CamControllerCallback) enum.nextElement();
     callback.addWebCam(location);
   }

// Notify MediaServer so it can update it's GUI
   sendAddWebCamEvents(location);
 }
```

```
  // Notify Server we're going down so Cam Controllers don't try to access us
  public synchronized void removeWebCam(String location) {
    // Remove web cam from list
    webCamList.remove(location);

    Enumeration enum = camControllers.elements();
    logger.info("Notifying all cam controllers to remove web cam: " + location);
    while (enum.hasMoreElements()) {
      CamControllerCallback callback = (CamControllerCallback) enum.nextElement();
      callback.removeWebCam(location);
    }

    // Notify MediaServer so it can update it's GUI
    sendRemoveWebCamEvents(location);
  }

  /**
   * Return Media Server's host name
   * @return The Server host name
   */
  public synchronized String getServerHostName() {
    String serverHostName = "";

    try {
      java.net.InetAddress ia = java.net.InetAddress.getLocalHost();
      serverHostName = ia.getHostName();
    }
    catch (java.net.UnknownHostException ex) {
      logger.error("",ex);
      //ex.printStackTrace();
    }

    return serverHostName;
  }

  /**
   * Return location names of active web cams, empty array of size 1 if none available
   * @return Location names of active web cams
   */
  public synchronized String[] getLocations() {
    String[] loc = {""};

    if (webCamList.size() != 0) {
      loc = new String[webCamList.size()];
      webCamList.copyInto(loc);
    }

    return loc;
  }

}
```

CamManagerImpl.java

```
/*******************************CamManagerImpl.java*******************************/

package camP;

import java.io.*;
import java.net.*;
import java.util.*;

import javax.media.protocol.*;
import javax.media.*;

import idlPack.*;
import camP.gui.CamProcessorFrame;
import camP.logging.LoggerFactory;
import camP.logging.Logger;


/**
 *  Object is created by a CamProcessor and used by a CamController
```

```
 * to control a web cam remotely.
 * <p>
 * @author Ciprian Baciu<br>
 * Technical University of Cluj-Napoca<br>
 * May 2004
 */

public class CamManagerImpl extends CamManagerPOA {

  /**
   * callbacks used for remote events
   */
  protected Hashtable callbacks;

  /**
   * callback for CamController comunication.
   */
  protected CamVideoCallback callback;

  /**
   * location of cam manager
   */
  private String camLocation;

  /**
   *
   */
  private RTPExport export;

  /**
   * The server host name
   */
  private String serverHostName = null;

  /**
   *  The temporary file creating when recording. This file is
   * transmitted to  the MediaServer after recording, and then
   * it is deleted.
   */
  private final String TEMP_VIDEO_FILENAME = "temp.mov";

  /**
   * Boolean state variable - true if transmitting videos
   */
  private boolean transmittingVideo = false;

  /**
   * Boolean state variable - true f recording videos
   */
  private boolean recordingVideo = false;

  private static Logger logger;

  /**
   * Creates a CamManager object on the <code>location</code> location
   * @param location The location of the CamManager
   */
  public CamManagerImpl(String location) {
    camLocation = location;

    export = new RTPExport();
    callbacks = new Hashtable();
    try {
      logger = LoggerFactory.getLogger(CamManagerImpl.class.getName(), "activity.log");
      logger.setLevel(CamProcessor.appLoggingLevel);
    }
    catch (IOException e) {
      e.printStackTrace();
    }

  }

  /**
   * Need server's host name to connect with socket
   * @param hostname The MediaServer host name
   */
```

```
 public void setServerName(String hostname) {
   serverHostName = hostname;
 }

 //|||||||||||||||||| Remote method implementations ||||||||||||||||||//

 /**
  *  Start transmitting live video stream to CamController at given ipAddr.
  * The callback is used for notifiying CamController when the transmission
  * has begun.
  * @param hostName The Ip address of the CamController
  * @param callback gets called when video is finished
  * @return false if already transmitting, true otherwise.
  */
 public synchronized boolean startTransmission(String hostName, CamVideoCallback callback) {
   logger.info("startTransmission ");

   if (transmittingVideo) {
     return false;
   }

   transmittingVideo = true;

   // The callback gets called when video is finished
   this.callback = callback;

   // Start sending video to ipAddr and notify receiver of incoming RTP transmission
   String url = "rtp://" + hostName + ":5050/video";

   logger.info("sending to " + url);

   CamProcessor.transmitVideo(hostName);

   callback.videoAvailable(url);

   return true;
 }

 /**
  * Stop transmitting live video stream to CamController.
  */
 public void stopTransmission() {
   logger.info("stopTransmission called");

   transmittingVideo = false;

   CamProcessor.stopTransmitVideo();
 }

 /**
  *  Start recording to file.  Return false if already busy recording to file,
  * true otherwise.
  * @param maxRecTime The maximum time for recording
  * @param callback gets called when video is finished
  * @return false if already recording
  */
 public boolean startRec(int maxRecTime, CamVideoCallback callback) {
   logger.info("startRec called");

   if (recordingVideo) {
     return false;
   }

   recordingVideo = true;

   // The callback gets called when video is finished
   this.callback = callback;

   // Start in new thread so we can return from this function immediately.  This allows
   // stopRec to be called.
   ExportThread et = new ExportThread(maxRecTime);
   et.start();

   logger.info("startRec finished.");

   return true;
```

```
 }

 /**
  * Send video file from CamProcessor to MediaServer using sockets.
  */
 private void transmitFile() {
   BufferedOutputStream fos = null;
   Socket socket = null;

   try {
     logger.info("Writing file to server socket: " + serverHostName + ":" +
         CamProcessor.FILE_SERVER_PORT);

     // open a socket connection.  Need server host name and socket
     socket = new Socket(serverHostName, CamProcessor.FILE_SERVER_PORT);

     // open I/O streams for objects
     fos = new BufferedOutputStream(socket.getOutputStream());

     File transferFile = new File(TEMP_VIDEO_FILENAME);
     BufferedInputStream fin = new BufferedInputStream(new FileInputStream(transferFile));

     // First write out location so Server can name file.
     String fileName = camLocation + "\n";
     byte byteArray[] = new byte[fileName.length()];
     byteArray = fileName.getBytes();

     fos.write(byteArray);

     int c = fin.read();
     while (c != -1) {
       fos.write(c);
       c = fin.read();
     }

     fin.close();
     fos.close();
   }
   catch (Exception e) {
     logger.error("Error transmitting file to Server.", e);
     //e.printStackTrace();
   }
 }

 /**
  * Stop recording video stream.
  */
  public void stopRec() {
   logger.info("stopRec called");

   export.stopExport();

   recordingVideo = false;
 }

 /**
  * Thread to export video to file, transmit to MediaServer, and notify CamController
  */
 class ExportThread extends Thread {

   /**
    * The maximum recording time
    */
   private int maxRecTime;

   /**
    * Creates an file export theread
    * @param maxRecTime The maximum recording time
    */
   ExportThread(int maxRecTime) {
     this.maxRecTime = maxRecTime;
   }

   public void run() {
     // Generate output media locators
```

```
      String currentDir = System.getProperty("user.dir");
      String fileName = currentDir + "/" + TEMP_VIDEO_FILENAME;

      MediaLocator oml = new MediaLocator("file:/" + fileName);

      logger.info("Starting RTPExport...");

      CamProcessorFrame.lblInfo.setText("Recording to file... ");

      // Must create clone of data source so video continues to play in CamProcessor.
      if (!export.startExport(((SourceCloneable) CamProcessor.dataSource).createClone(), oml, maxRecTime)) {
        logger.error("RTPExporting failed", null);
        VideoClip video = new VideoClip("", "", 0, 0, 0);
        callback.newRecording(video);
      }
      else {
        // Notify client of new media
        VideoClip video = new VideoClip();
        video.location = camLocation;
        video.timeStamp = System.currentTimeMillis();

        // Name is taken from timestamp.  Replace spaces and : with underscores
        video.name = new Date(video.timeStamp).toString();
        video.name = video.name.replace(' ', '_');
        video.name = video.name.replace(':', '_');

        File f = new File(TEMP_VIDEO_FILENAME);
        video.size = (int) f.length();

        video.length = export.actualRecTime; //maxRecTime;

        // Now that video is produced, send it to Server for storing using sockets

        CamProcessorFrame.lblInfo.setText("Transmitting to server... ");
        transmitFile();

        CamProcessorFrame.lblInfo.setText("");

        // Notify CamController of new video
        callback.newRecording(video);

        recordingVideo = false;
      }
    }
  }  // end ExportThread
}
```

CamVideoCallbackImpl.java

```
/**********************************CamVideoCallbackImpl.java**********************************/

package camC;


import idlPack.CamVideoCallbackPOA;
import idlPack.VideoClip;

import camC.logging.Logger;
import camC.logging.LoggerFactory;
import camC.gui.CamControllerFrame;

import java.io.IOException;


/**
 * Class for notification of new videos.
 */

public class CamVideoCallbackImpl extends CamVideoCallbackPOA {
  /**
   * The logger for this class
   */
  private static Logger logger;
```

```
  public CamVideoCallbackImpl() {
    try {
      logger = LoggerFactory.getLogger(CamControllerCallbackImpl.class.getName(), "activity.log");
      logger.setLevel(CamController.appLoggingLevel);
    }
    catch (IOException e) {
      e.printStackTrace();
    }

  }

  /**
   * Called by CamProcessor when video has finished recording to file and has been transmitted to the Server.  This
   * notifies the CamController of the new video so it can tell the server to store it in the database.
   */
  public void newRecording(VideoClip video) {
    // Right now if location is blank, there's a problem.  Could make more robust by
    // throwing an exception.  ORBacus doesn't like to send "null", so exceptions are the better way to go.
    if (video.location.equals("")) {
      logger.error("Error getting video from exporter.", null);
      CamControllerFrame.lblInfo.setText("Error getting video");
      CamController.recording = false;
      CamControllerFrame.showError("Error retrieving video.");
      return;
    }

    logger.info("Received new video " + video.name + " from location " + video.location);

    CamControllerFrame.lblInfo.setText("");
    CamController.recording = false;

    // The CamProcessor sends the video to the MediaServer via sockets but
    // does not tell the server to store the video since the MediaStore object
    // is not accessible to the CamProcessor.  It is only instantiated by the
    // CamController, and that is why it is responsible for telling the server
    // to store the video.

    // Store new video on the Server
    CamController.mediaStore.storeVideo(video);

    // Tell VideoArchivePanel to repopulate if this is a new archive location
    //archivePanel.updateLocationList(video.location);
  }

  /**
   * Notify CamController of video that is being streamed to the given URL.
   *
   * @param url
   */
  public void videoAvailable(String url) {
    logger.info("Video available at url: " + url);

    logger.info("isSelected=" + CamControllerFrame.watchCheckBox.isSelected());
    logger.info("videoPlaying=" + CamControllerFrame.videoPlaying);
    if (CamControllerFrame.watchCheckBox.isSelected() && CamControllerFrame.videoPlaying == false) {
      CamControllerFrame.videoPlaying = true;
      CamControllerFrame.videoPanel.play(url);
    }

    CamControllerFrame.lblInfo.setText("");
  }
} // End CamVideoCallbackImpl class
```

3. Proceduri stocate SQL Server

```
CREATE PROCEDURE [dbo].[ReadAll_media_By_location]
                @location varchar(50)
AS
SET NOCOUNT ON
SELECT
        [name],
        [size],
        [timestamp],
        [length]
FROM
        [media]
WHERE
        [location] = @location
GO


CREATE PROCEDURE [dbo].[ReadAll_webCams_By_name]
        @name varchar(50)
AS
SET NOCOUNT ON
SELECT
        [id]
FROM
        [webCams]
WHERE
        [name] = @name
GO


CREATE PROCEDURE [dbo].[Insert_videos]
        @id int output,
        @webCamID int,
        @name varchar(100),
        @size int,
        @timestamp bigint,
        @length int
AS
SET NOCOUNT ON
INSERT INTO videos
(
        [webCamID],
        [name],
        [size],
        [timestamp],
        [length]
)
VALUES
(
        @webCamID,
        @name,
        @size,
        @timestamp,
        @length
)

SET @id = @@IDENTITY
GO
```