ΑΝΩΤΑΤΟ ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΡΗΤΗΣ

Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Εφαρμοσμένης Πληροφορικής & Πολυμέσων

Πτυχιακή Εργασία

"Εφαρμογή επικοινωνίας μεταξύ χρηστών μέσω διαδικτύου με δυνατότητα λειτουργίας σε κατάσταση τηλεδιάσκεψης"

ΟΝΟΜΑΤΕΠΩΝΥΜΟ: **ΚΑΣΤΑΝΙΔΗΣ ΑΛΕΞΑΝΔΡΟΣ**

ΕΙΣΗΓΗΤΗΣ: ΑΘΑΝΑΣΙΟΣ Γ. ΜΑΛΑΜΟΣ

Περιεχόμενα

1. **Πρόλογος**

Αυτή η Εργασία είναι ένας τρόπος επικοινωνίας μεταξύ διαδικτύου πολλών χρηστών οι οποίοι μπορεί να είναι μια παρέα, μια οικογένεια, συνεργάτες, οι οποίοι θέλουν να μοιραστούν ο ένας κάτι με τον άλλο αισθήματα, ιδέες, απόψεις, αλλά δίνει την δυνατότητα τηλεδιάσκεψης ώστε ένας να μπορεί να βλέπει τον άλλο και να τον ακούει ίσως για κάποιες πιο προσωπικές στιγμές, ίσως για κάποιες ιδέες, σκέψεις οι οποίες έχουν μόνο κάποιον ως παραλήπτη, η για οποιονδήποτε άλλο λόγω τον οποίο βρίσκει ο κάθε χρήστης απαραίτητο.
Ο σύγχρονος άνθρωπος είναι πλέον πολύ απαιτητικός και έχει το δικαίωμα αυτό εφόσον η τεχνολογία έχει φτάσει σε ένα αρκετά υψηλό επίπεδο, έτσι αυτός ζητάει να μπορεί να έχει επαφή με οποιονδήποτε ,οποτεδήποτε χωρίς να τον ενδιαφέρουν τα προβλήματα υλικού, επικοινωνίας τεχνολογιών, η οτιδήποτε αφορά της τεχνολογίες και τα πρωτόκολλα που χρησιμοποιούνται για αυτές τις υπηρεσίες, θέλουν και πολλές φορές έχουν ανάγκη να επικοινωνούν χωρίς προβλήματα.
Επειδή δυστυχώς η ευτυχώς για τον καθένα πλέον οι επικοινωνία είναι μείζον θέμα, μπορείτε να δείτε από κινητά μέχρι PDA και μίνι Υπολογιστές έως και κανονικά Notebook τα οποία δεν αποχωριζόμαστε ποτέ για πολλούς λογούς αλλά και για την επικοινωνία μας.
Πλέον οι εταιρίες κινητής τηλεφωνίας έχουν όλες από κάποια πακέτα προσφορών για την χρήση του Διαδυκτίου μέσω κινητού τηλεφώνου μέχρι και για την χρήση συγκεκριμένων πολύ γνωστών προγραμμάτων (βλ. Messengers, Online TV).
Έτσι βλέπουμε ότι γίνεται όλο και πιο επιτακτική η επικοινωνία μέσω οποιουδήποτε υλικού ακόμη και λογισμικού, κινητά με χρήση ειδικών λογισμικών όπως Symbian η κινητά με χρήση λογισμικού Windows,
έτσι χρησιμοποιώντας την JAVA μπορούμε και έχουμε την ανεξαρτητοποίηση από το λογισμικό αλλά και με την χρήση του JMF μπορούμε αν ανεξαρτητοποιηθούμε παράλληλα από το υλικό λόγω της χρήσης μιας μεγάλης λίστας από Formats για την κωδικοποίηση του ήχου-βίντεο.
Ακόμη πλέον υπάρχουν προβλήματα επικοινωνίας συγκεκριμένων τεχνολογιών, λανθασμένη η ημιτελής επικοινωνία μεταξύ τεχνολογιών έως και μεγάλες ασυμβατότητες και δεν μας δίνουν τη δυνατότητα αλλά και σε ορισμένες περιπτώσεις μας απαγορεύουν τη χρήση κάποιων πρωτοκόλλων επικοινωνίας για λογούς που αναφέραμε παραπάνω (λογισμικό, υλικό).
Ακόμη και αυτό το πρόβλημα λύνεται με την χρήση της JAVA και του JMF.
Έτσι τα πλέον μείζονα προβλήματα στην επικοινωνία μας αρχίζουν να μικραίνουν έως και να εξαλείφονται.

2. Εισαγωγή

Σκοπός της εργασίας είναι η χρήση της γλωσσάς Προγραμματισμού JAVA μαζί με το Java Media Framework (JMF) για την επικοινωνία πραγματικού χρόνου και την τηλεδιάσκεψη με χρήση ήχου και βίντεο μεταξύ χρηστών.
Η εφαρμογή στηρίζεται σε μοντέλο client-server όπου η δρομολόγηση της πληροφορίας γίνεται από τον εξυπηρετητή, η πλήρης εξήγηση της οποίας θα υπάρξει παρακάτω.
Ακόμη υπάρχουν εργαλεία ελέγχου, συγχρονισμού με τα οποία μπορούμε οποιαδήποτε στιγμή να πάρουμε πληροφορίες αλλά και να αυξομειώσουμε ,ώστε να ελλατώσουμε οποιαδήποτε καθυστέρηση που ίσως να υπάρξει, και με αυτόν τον τρόπο να συγχρονίσουμε Ήχο-Βίντεο.
Παράλληλα υπάρχουν τρόποι για περαιτέρω έρευνα και καλύτερα αποτελέσματα χρησιμοποιώντας το Java Media Framework (JMF) μερικοί από τους οποίους θα αναπτυχθούν παρακάτω.
Το JMF είναι μια προσπάθεια για την απεικόνιση βίντεο αλλά και την καταγραφή και αναπαραγωγή ήχου χωρίς προβλήματα υλικού και ασυμβατότητας με αυτά αλλά και λογισμικού που είναι πολύ μείζον θέμα στην εποχή που ζούμε και όλα αυτά μέσω ενός μικρού πακέτου που μαζί με την JAVA σχεδόν εξαλείφει αυτά τα προβλήματα.
Η Διάρθρωση της Εργασίας θα περιλαμβάνει μια περιγραφή του κώδικα της εργασίας μαζί με την ανάλυση και μια Θεωρητική περιπλάνηση στην JMF , ένα User Manual το οποίο θα μπορεί ο κάθε χρήστης να συμβουλεύεται οποιαδήποτε στιγμή κάτι δεν είναι κατανοητό, αλλά και ένα Developer Manual το οποίο θα μπορούν να χρησιμοποιήσουν Developers, για ίσως καλύτερα αποτελέσματα, περισσότερες πληροφορίες, χρήση περισσοτέρων η και καλύτερων κλάσεων η μεθόδων, και γενικά Πειραματισμό.

3. Θεωρητικό Υπόβαθρο

Η JMF επιτρέπει στους προγραμματιστές οι οποίοι θέλουν να αναπτύξουν προγράμματα
της JAVA με Time-based πολυμέσα, αλλά πλέον (JMF 2.0) υποστηρίζει τη σύλληψη και αποθήκευση πολυμέσων, έλεγχο του τύπου επεξεργασίας που εκτελείται κατά τη διάρκεια της αναπαραγωγής ήχου και εκτέλεση της επεξεργασίας σε stream πολυμέσων. Ακόμη υπάρχει μια μεγάλη αποθήκη από plug-in της JMF ώστε να επιτρέπεται σε προηγμένους προγραμματιστές να αναπτύξουν και να προσαρμόσουν ευκολότερα την λειτουργία της JMF. Υποστηρίζει με τη σύλληψη και αποθήκευση των πολυμέσων μεγαλύτερη αυτονομία και επεξεργασία από πλευράς των προγραμματιστών, και παράλληλα με την ποικιλία των plug-in όπως αναφέραμε επιτρέπει την πιο εύκολη προσαρμογή και επέκταση.
Η JMF 2.0 είναι σε σχεδιασμένη με σκοπό:

- Να είναι εύκολη στον προγραμματισμό
- Να υποστηρίζει σύλληψη των πολυμέσων
- Να επιτρέπει τον προγραμματισμό streaming πολυμέσων και Conferencing εφαρμογών.
- Επιτρέπει σε προχωρημένους προγραμματιστές η ομάδα αυτών να εφαρμόσουν δικές τους λύσεις βασισμένες στην JMF και εύκολα να ενσωματώσουν νέα χαρακτηριστικά γνωρίσματα στο συγκεκριμένο Framework (JMF).
- Επιτρέπει πρόσβαση σε RAW δεδομένα πολυμέσων (εύκολη επεξεργασία)
- Επιτρέπει στους προγραμματιστές τη δημιουργία , απόκτηση κωδικοποιητών , επεξεργαστών διαφόρων ειδικών εφέ, ακωδικοποίητων και αρκετών ακόμα μέσω την μεγάλης ποικιλίας των plug-in του JMF.
- Την συμβατότητα με παλαιότερες εκδόσεις αλλά και πιο νέες εκδόσεις του Framework.
- Ακόμη υποστηρίζει την εκπομπή και λήψη πολυμέσων χρησιμοποιώντας τα πολύ γνωστά Real-Time Transport Protocol και Real-Time Control Protocol

Έτσι δίνει τη δυνατότητα χρήσης είτε Time-based είτε Real-Time πολυμεσικών εφαρμογών.

Εμείς θα ασχοληθούμε με το Real-Time (Streaming Media) και ότι περιλαμβάνει η JMF σε αυτό.

4. Real-Time Media Streams

Για να μπορούμε να στέλνουμε η να λαμβάνουμε ζωντανά πολυμέσα η να δημιουργήσουμε ένα video-Conference μέσω Διαδυκτίου η τοπικού δικτύου πρέπει να μπορούμε να λαμβάνουμε και να μεταδίδουμε πολυμεσικά ρεύματα (media streams) σε πραγματικό χρόνο.
Σε αυτό το κεφάλαιο εξηγούμε τη χρήση του Real-Time Transport Protocol που χρησιμοποιεί το JMF για λήψη και μετάδοση αυτών των streams μέσω δικτύων.

Streaming Media

Όταν χρησιμοποιούμε για επικοινωνία streams πολυμέσων σε πραγματικό χρόνο ο χρήστης ξεκινάει να λαμβάνει και να βλέπει το stream χωρίς να περιμένει να 'κατέβει' ολόκληρο για να το δει. Μάλιστα το stream αυτό δεν χρειάζεται να έχει κάποια προκαθορισμένη διάρκεια οπότε το κατέβασμα με σκοπό την αναπαραγωγή θα ήταν αδύνατο.
Η χρήση των streaming media (πολυμέσων) είναι πλέον παντού στο Διαδίκτυο ζωντανό ραδιόφωνο, τηλεόραση, κονσέρτα που εκπέμπουν στο Διαδίκτυο και οποιοσδήποτε μπορεί να συνδεθεί και να τα παρακολουθήσει και πολλά Ακόμη γεγονότα από διάφορες ιστοσελίδες. Έτσι βλέπουμε ότι με την χρήση των Streaming media είναι πλέον πιο εύκολη, πολύ πιο γρήγορη και δυνατή σε περιπτώσεις που θα ήταν αδύνατη (πολυμέσα χωρίς προκαθορισμένη διάρκεια) η επικοινωνία.

Πρωτοκόλλα Streaming Media

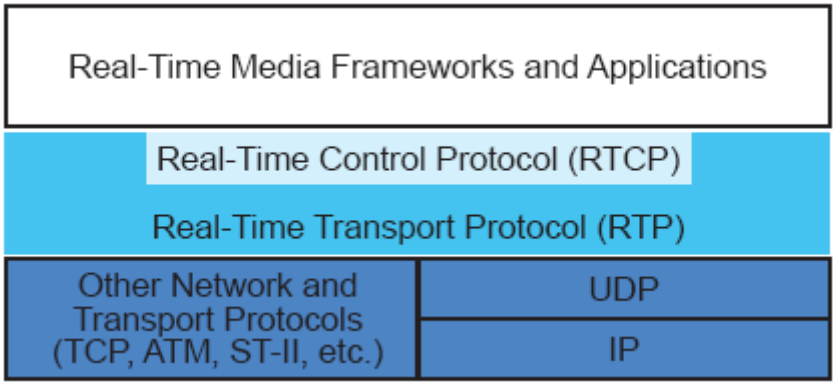
Η μετάδοση πολυμέσων μέσω Διαδυκτίου σε πραγματικό χρόνο σημαίνει αυτόματα και μεγάλες απαιτήσεις σε εύρος ζώνης με αλλά λόγια ταχύτητας για την κυριολεκτική έννοια της σε πραγματικό χρόνο εκπομπής πολυμέσων.
Είναι πιο εύκολη η αντιστάθμιση για χαμένα δεδομένα παρά για μεγάλες καθυστερήσεις στην λήψη δεδομένων. Υπάρχει μεγάλη διαφορά στην λήψη στατικών δεδομένων όπως αρχεία που το μόνο που μας ενδιαφέρει είναι η λήψη όλων των δεδομένων που περιέχει Έτσι τα Πρωτοκόλλα που χρησιμοποιούνται για στατικά δεδομένα δεν μπορούν να χρησιμοποιηθούν για streaming πολυμέσα.

Τα HTTP και FTP πρωτόκολλα χρησιμοποιούν το Transmission Control Protocol (TCP) που είναι πρωτόκολλο στο επίπεδο μεταφοράς και είναι σχεδιασμένο για αξιόπιστες μεταφορές δεδομένων σε μικρού εύρους ζώνης και μεγάλων σε πιθανότητα λάθη δικτύων. Όταν ένα πακέτο χαθεί η αλλοιωθεί επανεκπέμπεται.

Για αυτόν το λόγω αλλά Πρωτοκόλλα εκτός του TCP χρησιμοποιούνται για streaming πολυμέσα.
Ένα από αυτά που χρησιμοποιείται συχνά είναι το User Datagram Protocol (UDP). Το UDP είναι ένα μη αξιόπιστο πρωτόκολλο, δεν εγγυάται ότι κάθε πακέτο θα φτάσει στον προορισμό του , επιπλέον δεν υπάρχει καμία εγγύηση ότι τα πακέτα θα φτάσουν στην σειρά που στάλθηκαν Έτσι ο παραλήπτης πρέπει να είναι ικανός να αντισταθμίσει τα χαμένα δεδομένα , τα διπλά πακέτα, και τα πακέτα που έρχονται εκτός σειράς.
Το πρότυπο για την μετάδοση σε πραγματικό χρόνο δεδομένων όπως ήχος και βίντεο είναι το Real-Time Transport Protocol (RTP).

Real-Time Transport Protocol (RTP)

Το RTP περιλαμβάνει μια από άκρη σε άκρη δικτυακή παράδοση της μετάδοσης σε πραγματικό χρόνο δεδομένων. Είναι πρωτόκολλο Δικτύου και Μεταφοράς ανεξάρτητα αν και πολύ συχνά χρησιμοποιείται πάνω από το UDP.



Εικόνα 1

Το RTP πρωτόκολλο μπορεί να χρησιμοποιηθεί πάνω από μονής αλλά και από πολλαπλής διανομής δικτυακές υπηρεσίες.
Σε μονής διανομής υπηρεσίες ξεχωριστά αντίγραφα των δεδομένων εκπέμπονται από την πηγή σε κάθε παραλήπτη.
Σε πολλαπλής διανομής τα δεδομένα στέλνονται από την πηγή μια φορά και το δίκτυο είναι υπεύθυνο για την μετάδοση του στους παραλήπτες.
Η πολλαπλή διανομή είναι πιο αποδοτική για πολλές εφαρμογές όπως το video Conference. Το πρότυπο Internet Protocol (IP) υποστηρίζει πολλαπλή διανομή.

RTP Υπηρεσίες

Το RTP σου επιτρέπει την αναγνώριση του τύπου των δεδομένων που εκπέμπονται, καθορίζει την σειρά με την οποία τα πακέτα πρέπει να παρουσιαστούν και συγχρονίζει streams από διαφορετικές πηγές.

Τα RTP πακέτα δεν είναι εγγυημένο ότι θα φτάσουν με τη σειρά που στάλθηκαν Μάλιστα δεν είναι εγγυημένα ότι θα φτάσουν γενικά. Είναι στην πλευρά του παραλήπτη να πάρει και να ανακατασκευάσει τη σειρά των πακέτων και να ανιχνεύσει τα χαμένα πακέτα χρησιμοποιώντας της πληροφορίες της κεφαλίδας των πακέτων.

Όσο το RTP δεν παρέχει κανένα μηχανισμό για να κάνει σίγουρη την έγκαιρη μετάδοση η να παρέχει άλλους μηχανισμούς ποιότητας παροχής υπηρεσιών υποστηρίζεται από ένα πρωτόκολλο ελέγχου το RTCP που επιτρέπει την επίβλεψη της ποιότητας των πακέτων που έχουν διανεμηθεί, Ακόμη παρέχει έλεγχο και μηχανισμούς αναγνώρισης για τις εκπομπές RTP.

Αν σε κάποια εφαρμογή υπάρχει επιτακτική ανάγκη ποιότητας υπηρεσιών το RTP μπορεί να χρησιμοποιηθεί πάνω από ένα πρωτόκολλο που προσφέρει υπηρεσίες επικοινωνίας με σύνδεση.

Αρχιτεκτονική RTP

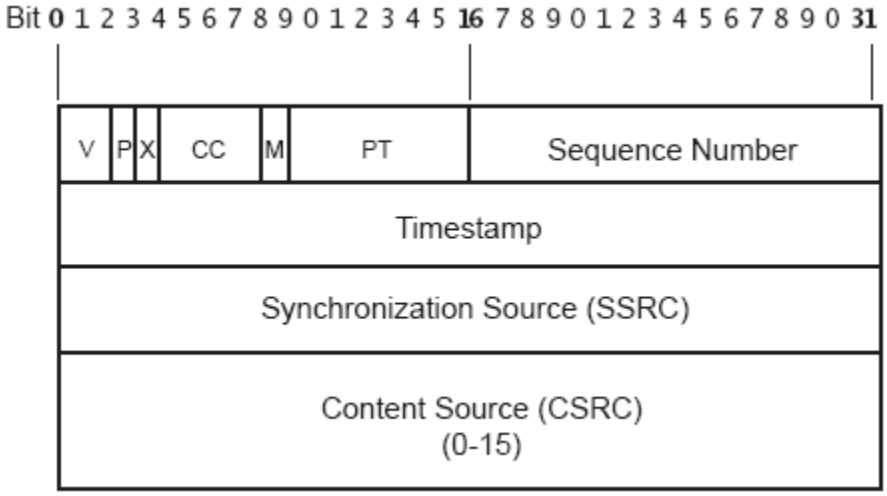
Μια RTP συνεδρία είναι μια ένωση από ένα σύνολο εφαρμογών που επικοινωνούν μαζί με το RTP. Μια συνεδρία αναγνωρίζεται από τη δικτυακή του διεύθυνση και ένα σύνολο πορτών. Μια πόρτα χρησιμοποιείται για τα πολυμεσικά δεδομένα και άλλη για το έλεγχο τους μέσω του RTCP.

Ένας συμμετέχων είναι ένα μηχάνημα, μια πηγή, η ένας απλός χρήστης που συμμετέχει στην συνεδρία.
Μια συμμετοχή σε μια συνεδρία μπορεί να σημαίνει παθητική λήψη δεδομένων (παραλήπτης) , ενεργητική εκπομπή δεδομένων (αποστολέας) η και τα δυο.

Κάθε τύπος πολυμέσων εκπέμπεται σε διαφορετική συνεδρία. Για παράδειγμα αν και τα δυο ήχος, βίντεο πρέπει να χρησιμοποιηθούν σε μια συνεδρία, τότε μια συνεδρία χρησιμοποιείται για τον Ήχο και άλλη ξεχωριστή για το βίντεο.
Αυτό επιτρέπει στους χρήστες να διαλέξουν τι είδος πολυμέσων θέλουν να λαμβάνουν, για παράδειγμα κάποιος με μικρό εύρος ζώνης μπορεί να θέλει να λαμβάνει μόνο Ήχο.

Πακέτα Δεδομένων

Τα πακέτα δεδομένων για μια συνεδρία εκπέμπονται σαν σειρά από πακέτα.
Μια σειρά πακέτων που προέρχονται από συγκεκριμένη πηγή έχουν το όνομα ως ένα RTP Stream (ρεύμα δεδομένων).
Κάθε πακέτο RTP μέσα σε ένα stream (ρεύμα) αποτελείται από δυο στοιχειά, μια κατασκευασμένη κεφαλίδα και τα κάθε αυτό δεδομένα .



Εικόνα 2

Η Κεφαλίδα ενός RTP πακέτου δεδομένων περιέχει:

- Ένα RTP νούμερο έκδοσης (V) : 2 bits το οποίο στην προκειμένη περίπτωση λόγω JMF 2.0 είναι το νούμερο 2.
- Padding (P) : 1 bit Αν αυτό το bit υφίσταται τότε υπάρχει ένα η περισσότερα bytes στο τέλος του πακέτου που δεν είναι μέρος του πακέτου.
- Extension (X) : 1 bit Αν αυτό το bit υφίσταται η κεφαλίδα ακολουθείται από Ακόμη μια προέκταση της κεφαλίδας. Αυτό χρησιμοποιείται για να μπορούμε να προσθέτουμε πληροφορίες στην κεφαλίδα.
- CSRC Count (CC) : 4 bits Το νούμερο αυτό αναγνωρίζει τι ακολουθεί την κεφαλίδα επέκτασης. Αν αυτό είναι μηδενικό τότε η πηγή συγχρονισμού είναι η ίδια με την πηγή του ωφέλιμου φορτίου.
- Marker (M) : 1 bit είναι ένα bit που προσδιορίζεται από τα προφίλ των πολυμέσων
- Payload Type (PT) : 7 bits Ένας δείκτης σε ένα πίνακα πολυμέσων που περιγράφει τον τύπο του ωφέλιμου φορτίου.
- Sequence Number :16 bits Ένα μοναδικό νούμερο πακέτου που χρησιμοποιείται στην αναγνώριση της σειράς που ανήκει το πακέτο. Το νούμερο αυξάνεται κατά ένα με κάθε νέο πακέτο.

- Timestamp: 32 bits Απεικονίζει τη δειγματοληψία του πρώτου Byte του ωφέλιμου φορτίου. Μεγάλος αριθμός συνεχόμενων πακέτων μπορεί να έχουν τον ίδιο αριθμό αφού μπορεί να προέρχονται από τον ίδιο video frame.
- SSRC:32 bits Αναγνωρίζει την πηγή συγχρονισμού. Αν το CSRC count είναι μηδενικό τότε η πηγή συγχρονισμού είναι η ίδια με αυτή του ωφέλιμου φορτίου αλλιώς αναγνωρίζει των mixer.
- CSRC : 32 bits each .Αναγνωρίζει τις πηγές που συνεισφέρουν στο ωφέλιμο φορτίο. Το νούμερο αυτό ενδεδειγνύεται από το CSRC count μπορούν να υπάρξουν έως και 16 πηγές. Αν υπάρχουν πολλές τότε το ωφέλιμο φορτίο είναι τα αναμειγνυόμενα δεδομένα από αυτές τις πηγές.

Πακέτα Ελέγχου

Επιπλέον στα πολυμεσικά δεδομένα μια συνεδρία, δεδομένα ελέγχου στέλνονται περιοδικά σε όλους τους συμμετέχοντες της συνεδρίας. Αυτά τα RTCP πακέτα μπορεί να περιλαμβάνουν πληροφορίες για την ποιότητα της υπηρεσίας των συμμετεχόντων της συνεδρίας, πληροφορίες για τις πηγές των πολυμεσικών δεδομένων που εκπέμπονται στις πόρτες δεδομένων αλλά και στατιστικά που αφορούν τα δεδομένα που έχουν εκπεμφθεί μέχρι τώρα.

Υπάρχουν αρκετοί τύποι από RTCP πακέτα :

- Αναφορά αποστολέα
- Αναφορά παραλήπτη
- Περιγραφή πηγής
- Αντίο
- Συγκεκριμένα πακέτα διαφορών εφαρμογών

Τα RTCP πακέτα είναι στοιβάδα και στέλνονται ως σύνθετα πακέτα που περιλαμβάνουν τουλάχιστον 2 πακέτα, ένα πακέτο αναφοράς και ένα πακέτο περιγραφής.

Όλοι οι συμμετέχοντες σε μια συνεδρία στέλνουν RTCP πακέτα. Ένας συμμετέχων που έχει στείλει πρόσφατα πακέτα δεδομένων εκδίδει μια αναφορά αποστολέα. Αυτή η αναφορά (SR) περιλαμβάνει το συνολικό αριθμό πακέτων και Byte που έστειλε όπως και πληροφορίες που μπορούν να χρησιμοποιηθούν για συγχρονισμό των πολυμεσικών streams από διαφορετικές συνεδρίες.

Οι συμμετέχοντες με μια συνεδρία εκδίδουν και μια αναφορά παραλήπτη για όλες τις πηγές που λαμβάνουν αυτοί πακέτα. Μια αναφορά παραλήπτη (RR) περιλαμβάνει πληροφορίες για τον αριθμό των πακέτων που χάθηκαν, το μεγαλύτερο νούμερο ακολουθίας που λήφθηκε και ένα χρονικό γραμματόσημο που μπορεί να χρησιμοποιηθεί για μια εκτίμηση της καθυστέρησης μεταξύ αποστολέα και παραλήπτη.

Το πρώτο πακέτο σε μια στοιβάδα RTCP πακέτου πρέπει να είναι ένα πακέτο αναφοράς Ακόμη και αν δεν έχουν σταλθεί η παραληφθεί πακέτα.

Όλες οι στοιβάδες RTCP πακέτων πρέπει να περιλαμβάνουν ένα στοιχείο περιγραφής της πηγής (SDES) που να περιέχει ένα όνομα (CNAME) που να αναγνωρίζει την πηγή. Περαιτέρω πληροφορίες μπορεί να υπάρχουν στην περιγραφή της πηγής όπως το όνομα της πηγής, η διεύθυνση email , τηλέφωνο, γεωγραφική τοποθεσία, όνομα εφαρμογής, η ένα μήνυμα που να περιγράφει την κατάσταση της πηγής.

Όταν μια πηγή δεν είναι πλέον ενεργή στέλνει ένα RTCP ANTIO (BYE) πακέτο, αυτό το πακέτο μπορεί να περιλαμβάνει το λόγω που η πηγή φεύγει από τη συνεδρία.

RTCP APP πακέτα δίνουν ένα μηχανισμό στις εφαρμογές για να καθορίσουν και να στέλνουν πληροφορίες μέσω της πόρτας ελέγχου RTP.

RTP Εφαρμογές

ΟΙ εφαρμογές RTP πολύ συχνά χωρίζονται σε αυτές που πρέπει να είναι ικανές να λαμβάνουν δεδομένα από το δίκτυο (RTP πελάτες) και αυτούς που πρέπει να είναι ικανοί να μεταδίδουν δεδομένα στο δίκτυο (RTP Servers). Μερικές εφαρμογές και τα δυο, για παράδειγμα οι εφαρμογές Conference λαμβάνουν και εκπέμπουν δεδομένα ταυτόχρονα από το δίκτυο.

Λαμβάνοντας Πολυμεσικά Streams

Η ικανότητα να δεχόμαστε RTP streams είναι απαραίτητο πλέον σε μια πληθώρα εφαρμογών. Για παράδειγμα:

- Εφαρμογές Conference πρέπει να είμαστε ικανοί να δεχόμαστε RTP streams και να τα προβάλουμε.

- Ένας τηλεφωνητής πρέπει να είναι ικανός να δέχεται RTP streams και να τα αποθηκεύει σε ένα αρχείο.
- Μια εφαρμογή που ηχογραφεί συζητήσεις πρέπει να είναι ικανή να λαμβάνει RTP streams και ταυτόχρονα να τα αποθηκεύει σε αρχείο και να τα προβάλει.

Εκπέμποντας Πολυμεσικά Streams

Οι λεγόμενοι RTP servers εκπέμπουν αποθηκεμένα η πολυμεσικά δεδομένα που έχουν συλλάβει από κάποια συσκευή στο δίκτυο.

Για παράδειγμα σε μια εφαρμογή Conference (τηλεδιάσκεψης) ένα RTP stream μπορεί να μεταδοθεί είτε από μια κάμερα που μόλις έχουμε συλλάβει το βίντεο , Ακόμη μπορεί τα δεδομένα να κωδικοποιηθούν σε διάφορα format και μετά να σταλθούν.

5. Κώδικας και ανάλυση

Η εργασία χωρίζεται στον server από τον οποίο γίνονται και οι μεταφορές μηνυμάτων , αυτό ώστε αν επιθυμούμε να κάνουμε έλεγχο τι ειπώθηκε ή και σε περίπτωση χρήσης από νέους σε ηλικία κάποια χρήση φίλτρων για συγκεκριμένες λέξεις, ο οποίος δεν περιέχει γραφικό περιβάλλον το οποίο ούτως άλλως δεν θα ήταν χρήσιμο (όσο ελαφρύτερος ο server τόσο το καλύτερο),και από τον client ο οποίος περιλαμβάνει γραφικό περιβάλλον, ένα κεντρικό JFrame στο οποίο έχουμε μια λίστα με τους χρήστες, μια περιοχή που εμφανίζονται τα μηνύματα, μια περιοχή που γράφουμε το μήνυμα μας, και μια μπάρα μενού οπού χρησιμοποιούμε για την ενεργοποίηση του Audio-Video ξεχωριστά η/και ταυτόχρονα, ένα JMenuItem για την έξοδο, και ένα JMenuItem για πληροφορίες του προγράμματος (About).
Ακόμη αφότου επιλέξουμε το χρήστη που θα μιλήσουμε private ανοίγει ένα Ακόμη JFrame που περιέχει την προσωπικη πλέον συζήτηση, τα πλαίσια Βίντεο, και τους ρυθμιστές του Audio/Video Buffer που μπορούμε αναλόγως να χειριστούμε.
Παρακάτω θα υπάρξει ανάλυση όλου του κώδικα με προσεκτικές πληροφορίες στην επικοινωνία Ήχου και Βίντεο με τη χρήση του JMF.

Κώδικας Server και Ανάλυση:

<**Αρχή κώδικα JChatServer**>

```
package com.jchat.server;

import com.jchat.server.JChatHandler;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Vector;

public class JChatServer {

   private static int nickCounter = 0;
   private static Vector chatHandlers = new Vector();

   public JChatServer(int port) throws Exception {
      ServerSocket server = new ServerSocket(port);
      System.out.println("[" + new java.util.Date() + "] Chat server started..");
      System.out.println("Server listening on Port [ " + port + " ]");
      while (true) {
         try {
            Socket client = server.accept();
```

```
            System.out.println("[" + new java.util.Date() + "] Accepted from " +
client.getInetAddress());
            JChatHandler chatHandler = new JChatHandler(client);
            chatHandler.start();
          } catch (Exception ex) {
            ex.printStackTrace();
          }
        }
    }

    public static void main(String args[]) throws Exception {
        JChatServer server = new JChatServer(9999);
    }

    public static String getNewClientNickname(){
        return "Guest (" + (++nickCounter) + ")";
    }

    public static Vector getChatHandlers(){
        return chatHandlers;
    }

    public static void addChatHandler(JChatHandler chatHandler){
        chatHandlers.add(chatHandler);
    }

    public static JChatHandler getChatHandler(String nickname){
        for(int i=0; i<chatHandlers.size(); i++){
            JChatHandler chatHandler = (JChatHandler) chatHandlers.get(i);
            if(chatHandler.getNickname().equalsIgnoreCase(nickname)){
                return chatHandler;
            }
        }
        return null;
    }
}
```

<**Τελος κωδικα**>

<**Αρχη κωδικα JChatHandler**>

```
package com.jchat.server;

import com.jchat.Constants;
import com.jchat.client.JChatClientInterface;
import java.net.*;
```

```
import java.io.*;
import java.util.*;

public class JChatHandler extends Thread {

   protected Socket socket;
   protected DataInputStream dataInput;
   protected DataOutputStream dataOutput;
   protected JChatClientInterface chatClient;
   protected String nickname = null;
   protected String ipAddress = null;

   public JChatHandler(Socket s) throws IOException {
      this.socket = s;
      dataInput = new DataInputStream(new
BufferedInputStream(s.getInputStream()));
      dataOutput = new DataOutputStream(new
BufferedOutputStream(s.getOutputStream()));
   }

   public String getIpAddress() {
      return ipAddress;
   }

   public String getNickname() {
      return nickname;
   }

   public void setNickname(String nickname) {
      this.nickname = nickname;
   }

   public void run() {
      ipAddress = socket.getInetAddress().toString();
      try {
         while (true) {
            String msg = dataInput.readUTF();
            processMessage(msg);
         }
      } catch (IOException ex) {
         broadcast(Constants.SERVER_MESSAGE_LEFT_NOTIFICATION + " "
+ nickname + "," + ipAddress);
         ex.printStackTrace();
```

```
      } catch (Exception ex1) {
         ex1.printStackTrace();
         broadcast(Constants.SERVER_MESSAGE_LEFT_NOTIFICATION + " "
+ nickname + "," + ipAddress);
      } finally {
         JChatServer.getChatHandlers().removeElement(this);
         try {
            socket.close();
         } catch (IOException ex) {
            ex.printStackTrace();
         }
      }
   }

   protected void processMessage(String message) {
      try {
         if (message.startsWith(Constants.CLIENT_MESSAGE_JOIN)) {
            String reqeustedNick =
message.substring(Constants.CLIENT_MESSAGE_JOIN.length() + 1);
            Enumeration e = JChatServer.getChatHandlers().elements();
            while (e.hasMoreElements()) {
               JChatHandler aHandler = (JChatHandler) e.nextElement();
               if (aHandler.getNickname().equalsIgnoreCase(reqeustedNick.trim()))
{
                  nickname = JChatServer.getNewClientNickname();
                  JChatServer.addChatHandler(this);

broadcast(Constants.SERVER_MESSAGE_JOIN_NOTIFICATION + " " +
nickname + ";" + ipAddress);
                  unicast(aHandler, Constants.SERVER_MESSAGE_CLIENT_LIST
+ " " + getClientList());
                  return;
               }
            }
            nickname = reqeustedNick;
            JChatServer.addChatHandler(this);
            broadcast(Constants.SERVER_MESSAGE_JOIN_NOTIFICATION + "
" + nickname + ";" + ipAddress);
            unicast(this, Constants.SERVER_MESSAGE_CLIENT_LIST + " " +
getClientList());
            return;
         } else if (message.startsWith(Constants.CLIENT_MESSAGE)) {
```

```
            broadcast(Constants.CLIENT_MESSAGE + " " + nickname + " says > "
+ message.substring(Constants.CLIENT_MESSAGE.length() + 1));
         } else if (message.startsWith(Constants.CLIENT_PRIVATE_MESSAGE))
{
            String messageContents =
message.substring(Constants.CLIENT_PRIVATE_MESSAGE.length() + 1);
            String msgComps[] = messageContents.split(";");
            String to = msgComps[0];
            String from = msgComps[1];
            String text = msgComps[2];
            JChatHandler aHandler = JChatServer.getChatHandler(to);
            if(aHandler != null){
               unicast(aHandler, message);
            } else {
               System.out.println("Un identified message : " + message);
            }
         }
      } catch (Exception ex) {
         ex.printStackTrace();
      }
   }

   protected String getClientList() {
      String clientList = "";
      for (int i = 0; i < JChatServer.getChatHandlers().size(); i++) {
         JChatHandler handler = (JChatHandler)
JChatServer.getChatHandlers().get(i);
         clientList += handler.getNickname() + "," + handler.getIpAddress() + (i <
(JChatServer.getChatHandlers().size() - 1) ? ";" : "");
      }
      return clientList;
   }

   protected void broadcast(String message) {
      Enumeration e = JChatServer.getChatHandlers().elements();
      while (e.hasMoreElements()) {
         JChatHandler handler = (JChatHandler) e.nextElement();
         try {
            unicast(handler, message);
         } catch (Exception ex) {
            ex.printStackTrace();
            handler.stop();
         }
```

```
        }
    }

    protected void unicast(JChatHandler handler, String message) {
        try {
            synchronized (dataOutput) {
                handler.dataOutput.writeUTF(message);
            }
            handler.dataOutput.flush();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

<**Τέλος κωδικα**>

Ανάλυση Κώδικα Server

Η JChatServer είναι η κεντρική Κλάση την οποία χρησιμοποιούμε για την λειτουργία του Server μας η οποία δημιουργεί ένα ServerSocket σε μια πόρτα (9999), και με την server.accept αρχίζει να ακούει για συνδέσεις μέσα από το δίκτυο-Διαδίκτυο, λίγο πιο κάτω με τη χρήση της JChatHandler chatHandler = new JChatHandler(client); Ξεκινάμε τον Handler του Server μας ο οποίος μέσα στον Constructor του δημιουργεί τα Input και Output streams για την μετακίνηση του κειμένου που θα χρησιμοποιήσει μετά ο client, το συγκεκριμένο Buffer Input/output stream είναι το κατάλληλο γιατί μπορεί να αδειάζει και να γεμίζει αναλόγως και δεν είναι στατικό κάτι που μας ενδιαφέρει πολύ διότι μπορεί να έχουμε μικρές η μεγάλες προτάσεις, και ακριβώς από κάτω Ξεκινάμε τον Handler όπου με την συγκεκριμένη εντολή πηγαίνουμε στην μέθοδο run του JChatHandler και εκεί διαβάζουμε την Ip όποιου χρήστη θέλει να συνδεθεί ,διαβάζουμε το εισερχόμενο stream και μέσω της συνάρτησης proccessMessage, μέσω της Constants όπου περιέχει κάποια String τα οποία χρησιμοποιούμε συχνά, χειριζόμαστε το μήνυμα (msg).

Εδώ Τελειώνει ο Server της Εργασίας και ξεκινάει ο Client.

Κώδικας Client και Ανάλυση :

<**Αρχή κώδικα JChatClient**>

```
package com.jchat;
```

```
import com.jchat.client.ClientProfile;
import com.jchat.client.JChatClientInterface;
import com.jchat.gui.FrmJChatClient;
import com.jchat.utils.IPAddressUtility;
import java.net.InetAddress;
import java.net.Socket;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.UIManager;

public class JChatClient {

   private static FrmJChatClient frameChatClient;
   private static ClientProfile myProfile = new ClientProfile();

   public static void main(String[] args) {
      try {
         if(args.length <= 0 || args.length > 1){
            System.err.println("Please mention the Server IP by modifying
client.bat");
            System.exit(1);
         }

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
         Socket s = new Socket(args[0], 9999);
         myProfile.setIpAddress(IPAddressUtility.getLocalIPAddress());
         JChatClientInterface chatClient = new
JChatClientInterface(s.getInputStream(), s.getOutputStream());
         frameChatClient = new FrmJChatClient(chatClient);
         if (myProfile.getNickname() == null ||
myProfile.getNickname().equalsIgnoreCase("")) {
            String nickname =
JOptionPane.showInputDialog(JChatClient.getChatFrame(), "Enter your
Nickname: ", "Required Information",
JOptionPane.INFORMATION_MESSAGE);
            myProfile.setNickname(nickname == null ? "" : nickname);
         }
         chatClient.startChat();
         frameChatClient.setTitle(Constants.APP_TITLE);
         frameChatClient.setSize(750, 500);
         com.jchat.utils.SwingUtilities.centerFrame(frameChatClient);
         frameChatClient.setVisible(true);
         frameChatClient.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    public static FrmJChatClient getChatFrame() {
        return frameChatClient;
    }

    public static ClientProfile getMyProfile() {
        return myProfile;
    }
}
```

<**Τέλος Κώδικα**>

Χρησιμοποιούμε τη JChatClient για να ανοίξουμε τον client οπού εκεί αφού δημιουργήσουμε το Socket με την IP που έχουμε πάρει και αφού βάλουμε το nickname που θέλουμε, στέλνουμε το JOIN στο server ούτως ώστε να μας προσθέσει στη λίστα με τους χρήστες, ανοίγει το κεντρικό JFrame έχουμε το λειτουργικό client μας.

<**Αρχή Κώδικα JChatClientInterface**>

```
package com.jchat.client;

import com.jchat.Constants;
import com.jchat.JChatClient;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Vector;

public class JChatClientInterface implements Runnable {

    private Thread listener = null;
    private DataInputStream dataInput = null;
    private DataOutputStream dataOutput = null;

    public JChatClientInterface(InputStream is, OutputStream os) {
        dataInput = new DataInputStream(is);
```

```
        dataOutput = new DataOutputStream(os);
        listener = new Thread(this);
    }

    public void startChat() {
        listener.start();
    }

    public void run() {
        sendChat(Constants.CLIENT_MESSAGE_JOIN + " " +
JChatClient.getMyProfile());
        try {
            while (true) {
                String line = dataInput.readUTF();
                processMessage(line);
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        } finally {
            listener = null;
            try {
                dataOutput.close();
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
    }

    private void processMessage(String message) {
        System.out.println("Processing : " + message);
        if
(message.startsWith(Constants.SERVER_MESSAGE_JOIN_NOTIFICATION)) {
            String filteredMessage =
message.substring(Constants.SERVER_MESSAGE_JOIN_NOTIFICATION.leng
th() + 1);
            String messageParts[] = filteredMessage.split(";");
            ClientProfile aProfile = new ClientProfile();
            aProfile.setNickname(messageParts[0]);
            aProfile.setIpAddress(messageParts[1]);
            JChatClient.getChatFrame().addClientProfile(aProfile);
            JChatClient.getChatFrame().updateChatWindow("** " + aProfile + " **
has Joined Chat..");
```

```
      } else if
(message.startsWith(Constants.SERVER_MESSAGE_CLIENT_LIST)) {
         try {
            Vector chattersList = new Vector();
            message =
message.substring(Constants.SERVER_MESSAGE_CLIENT_LIST.length() + 1);
            String[] chatters = message.split(";");
            for (int i = 0; i < chatters.length; i++) {
               String chatterProfile[] = chatters[i].split(",");
               ClientProfile aClient = new ClientProfile();
               aClient.setNickname(chatterProfile[0]);
               aClient.setIpAddress(chatterProfile[1]);
               chattersList.add(aClient);
            }
            JChatClient.getChatFrame().updateChattersList(chattersList);
         } catch (Exception ex) {
            ex.printStackTrace();
         }
      } else if
(message.startsWith(Constants.SERVER_MESSAGE_LEFT_NOTIFICATION))
{
         message =
message.substring(Constants.SERVER_MESSAGE_LEFT_NOTIFICATION.len
gth() + 1);
         String chatterProfile[] = message.split(",");
         JChatClient.getChatFrame().updateChatWindow("** " + chatterProfile[0]
+ " ** has left..");
         JChatClient.getChatFrame().removeChatter(chatterProfile[0],
chatterProfile[1]);
      } else if (message.startsWith(Constants.CLIENT_MESSAGE)) {

JChatClient.getChatFrame().updateChatWindow(message.substring(Constants.CL
IENT_MESSAGE.length() + 1));
      } else if (message.startsWith(Constants.CLIENT_PRIVATE_MESSAGE)){
         message =
message.substring(Constants.CLIENT_PRIVATE_MESSAGE.length() + 1);
         String msgComps[] = message.split(";");
         String to = msgComps[0];
         String from = msgComps[1];
         String text = msgComps[2];
         JChatClient.getChatFrame().deliverPrivateMessage(to, from, text);
      }
   }
```

```
    public void sendChat(String text) {
        try {
            dataOutput.writeUTF(text);
            dataOutput.flush();
        } catch (IOException ex) {
            ex.printStackTrace();
            listener.stop();
        }
    }
}
```
<**Τέλος κώδικα**>

Την κλάση αυτή την καλεί ο JChatClient ώστε να σετάρουμε τα Streams, και να περάσουμε στο ClientProfile τα στοιχεία μας. Μετά τη χρήση του start αφού κάνει implement Runnable τρέχει τη run όπου εκεί στέλνει την πρώτη φορά το JOIN για την προσθέσει μας στη λίστα και Μετά απλά διαβάζει συνέχεια το stream και ανάλογα το μήνυμα μέσω της proccessMessage αναλύει το μήνυμα και πράττει αναλόγως.

<**Αρχή κώδικα frmJChatClient**>

```
package com.jchat.gui;

import com.jchat.Constants;
import com.jchat.JChatClient;
import com.jchat.client.ClientProfile;
import com.jchat.client.JChatClientInterface;
import com.jchat.test.trasmitter.JChatAudioTransmitter;
import com.jchat.test.trasmitter.JChatVideoTransmitter;
import com.jchat.test.trasmitter.Target;
import java.util.Vector;
import javax.swing.DefaultListModel;
import javax.swing.JOptionPane;

public class FrmJChatClient extends javax.swing.JFrame {

    /** Creates new form FrmJChatClient */
    private JChatClientInterface chatClient = null;
    private static Vector privateChatWindows = new Vector();

    public FrmJChatClient(JChatClientInterface chatClient) {
        initComponents();
```

```
        this.chatClient = chatClient;
    }

    public JChatClientInterface getChatClient() {
        return chatClient;
    }

   public void updateChatWindow(String text) {
        txtChatWindow.append(text + "\n");
        this.requestFocus();
    }

    public void updateChattersList(Vector chattersList) {
        listModel.removeAllElements();
        for (int i = 0; i < chattersList.size(); i++) {
            listModel.addElement(chattersList.get(i));
        }
    }

    public void removeChatter(String nickname, String ipAddress) {
        for (int i = 0; i < listModel.size(); i++) {
            ClientProfile aClient = (ClientProfile) listModel.getElementAt(i);
            if (aClient.getNickname().equalsIgnoreCase(nickname) &&
aClient.getIpAddress().equalsIgnoreCase(ipAddress)) {
                listModel.removeElementAt(i);
                break;
            }
        }
    }

    public void addClientProfile(ClientProfile aProfile) {
        ((DefaultListModel) lstFriendsList.getModel()).addElement(aProfile);
    }

    public ClientProfile getClientProfile(String nickname) {
        try {
            DefaultListModel model = (DefaultListModel) lstFriendsList.getModel();
            for (int i = 0; i < model.size(); i++) {
                ClientProfile aProfile = (ClientProfile) model.get(i);
                if (aProfile.getNickname().equalsIgnoreCase(nickname)) {
                    return aProfile;
                }
            }
```

```
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return null;
    }

    public static Vector getPrivateChatWindows() {
        return privateChatWindows;
    }

    public static void setPrivateChatWindows(Vector privateChatWindows) {
        FrmJChatClient.privateChatWindows = privateChatWindows;
    }


    public void addPrivateChatWindow(FrmPrivateChat chatWindow) {
        if (!privateChatWindows.contains(chatWindow)) {
            privateChatWindows.add(chatWindow);
        }
    }

    public FrmPrivateChat getPrivatechatWindow(String nickname) {
        for (int i = 0; i < privateChatWindows.size(); i++) {
            FrmPrivateChat privateChatWindow = (FrmPrivateChat)
privateChatWindows.get(i);
            if
(privateChatWindow.getChatterProfile().getNickname().equalsIgnoreCase(nickna
me)) {
                return privateChatWindow;
            }
        }
        return null;
    }

    public void deliverPrivateMessage(String to, String from, String text) {
        FrmPrivateChat privateChat = null;
        privateChat = getPrivatechatWindow(from);
        if (privateChat != null) {

privateChat.updateChatWindow(privateChat.getChatterProfile().getNickname() +
" says > " + text);
        } else {
            ClientProfile chatterProfile = getClientProfile(from);
```

```
            if (chatterProfile != null) {
                privateChat = new FrmPrivateChat(chatterProfile);
                privateChat.setSize(600, 500);
                com.jchat.utils.SwingUtilities.centerFrame(privateChat);
                privateChat.setVisible(true);
                privateChatWindows.add(privateChat);

privateChat.updateChatWindow(privateChat.getChatterProfile().getNickname() +
" says > " + text);
            } else {
                System.out.println("Un identified chat message....");
            }
        }
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
        java.awt.GridBagConstraints gridBagConstraints;

        pnlMain = new javax.swing.JPanel();
        pnlChatInput = new javax.swing.JPanel();
        pnlControls = new javax.swing.JPanel();
        btnSendChat = new javax.swing.JButton();
        scrollPaneInput = new javax.swing.JScrollPane();
        txtChatInput = new javax.swing.JTextPane();
        splitPane = new javax.swing.JSplitPane();
        scrollPaneChatWindow = new javax.swing.JScrollPane();
        txtChatWindow = new javax.swing.JTextArea();
        scrollPaneFriendsList = new javax.swing.JScrollPane();
        lstFriendsList = new javax.swing.JList();
        mnuBar = new javax.swing.JMenuBar();
        mnuFile = new javax.swing.JMenu();
        mnuItemExit = new javax.swing.JMenuItem();
        mnuTools = new javax.swing.JMenu();
        mnuItemStartWebcam = new javax.swing.JMenuItem();
        mnuItemStartAudioStream = new javax.swing.JMenuItem();
        mnuHelp = new javax.swing.JMenu();
        mnuItemAbout = new javax.swing.JMenuItem();
```

```
setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        pnlMain.setLayout(new java.awt.BorderLayout());

        pnlChatInput.setBorder(javax.swing.BorderFactory.createTitledBorder("Chat
Input"));
        pnlChatInput.setLayout(new java.awt.BorderLayout());

        pnlControls.setLayout(new java.awt.GridBagLayout());


btnSendChat.setText("<html><body><center><H4>Send<br/>Chat</center></H4
></body></html>");
        btnSendChat.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                btnSendChatActionPerformed(evt);
            }
        });
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.insets = new java.awt.Insets(10, 10, 10, 10);
        pnlControls.add(btnSendChat, gridBagConstraints);

        pnlChatInput.add(pnlControls, java.awt.BorderLayout.EAST);

        scrollPaneInput.setViewportView(txtChatInput);

        pnlChatInput.add(scrollPaneInput, java.awt.BorderLayout.CENTER);

        pnlMain.add(pnlChatInput, java.awt.BorderLayout.SOUTH);

        splitPane.setDividerLocation(600);

        txtChatWindow.setColumns(50);
        txtChatWindow.setEditable(false);
        txtChatWindow.setLineWrap(true);
        txtChatWindow.setRows(5);
        scrollPaneChatWindow.setViewportView(txtChatWindow);

        splitPane.setLeftComponent(scrollPaneChatWindow);
```

```
        lstFriendsList.setModel(listModel);
        lstFriendsList.addListSelectionListener(new
javax.swing.event.ListSelectionListener() {
            public void valueChanged(javax.swing.event.ListSelectionEvent evt) {
                lstFriendsListValueChanged(evt);
            }
        });
        scrollPaneFriendsList.setViewportView(lstFriendsList);

        splitPane.setRightComponent(scrollPaneFriendsList);

        pnlMain.add(splitPane, java.awt.BorderLayout.CENTER);

        getContentPane().add(pnlMain, java.awt.BorderLayout.CENTER);

        mnuFile.setText("File");

        mnuItemExit.setText("Exit");
        mnuItemExit.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                mnuItemExitActionPerformed(evt);
            }
        });
        mnuFile.add(mnuItemExit);

        mnuBar.add(mnuFile);

        mnuTools.setText("Tools");

        mnuItemStartWebcam.setText("Start My Webcam");
        mnuItemStartWebcam.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                mnuItemStartWebcamActionPerformed(evt);
            }
        });
        mnuTools.add(mnuItemStartWebcam);

        mnuItemStartAudioStream.setText("Start Audio Stream");
        mnuItemStartAudioStream.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                mnuItemStartAudioStreamActionPerformed(evt);
```

```
            }
        });
        mnuTools.add(mnuItemStartAudioStream);

        mnuBar.add(mnuTools);

        mnuHelp.setText("Help");

        mnuItemAbout.setText("About");
        mnuItemAbout.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                mnuItemAboutActionPerformed(evt);
            }
        });
        mnuHelp.add(mnuItemAbout);

        mnuBar.add(mnuHelp);

        setJMenuBar(mnuBar);

        pack();
    }// </editor-fold>
    private void mnuItemExitActionPerformed(java.awt.event.ActionEvent evt) {
        if (com.jchat.utils.SwingUtilities.showWarnMessage("Do you really want to
exit?") == JOptionPane.OK_OPTION) {
            System.exit(0);
        }
    }

    private void btnSendChatActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            chatClient.sendChat(Constants.CLIENT_MESSAGE + " " +
txtChatInput.getText());
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        txtChatInput.setText("");
}

    private void lstFriendsListValueChanged(javax.swing.event.ListSelectionEvent
evt) {
        if (!evt.getValueIsAdjusting()) {
            if (lstFriendsList.getSelectedIndex() != -1) {
```

```
            ClientProfile aProfile = (ClientProfile)
listModel.getElementAt(lstFriendsList.getSelectedIndex());
            if (!aProfile.equals(JChatClient.getMyProfile())) {
               if (JChatClient.getMyProfile().isBusy()) {
                  com.jchat.utils.SwingUtilities.showErrorMessage("You can create
only one private chat session at a time. \nPlease try again after closing the current
private session.");
                  return;
               }
               for (int i = 0; i < privateChatWindows.size(); i++) {
                  FrmPrivateChat privateChat = (FrmPrivateChat)
privateChatWindows.get(i);
                  if
(privateChat.getChatterProfile().getNickname().equalsIgnoreCase(aProfile.getNic
kname())) {
                     privateChat.setSize(600, 500);
                     com.jchat.utils.SwingUtilities.centerFrame(privateChat);
                     privateChat.setVisible(true);
                     return;
                  }
               }
               FrmPrivateChat frmPrivateChat = new FrmPrivateChat(aProfile);
               frmPrivateChat.setSize(600, 500);
               com.jchat.utils.SwingUtilities.centerFrame(frmPrivateChat);
               frmPrivateChat.setVisible(true);
               addPrivateChatWindow(frmPrivateChat);
            }
         }
      }
   }

   private void
mnuItemStartWebcamActionPerformed(java.awt.event.ActionEvent evt) {
      if (mnuItemStartWebcam.getText().startsWith("Start")) {
         if (!JChatVideoTransmitter.getInstance().isRunning()) {
            Vector targets = new Vector();
            Target aTraget = new Target("2222",
JChatClient.getMyProfile().getIpAddress(), "4444");
            targets.add(aTraget);
            JChatVideoTransmitter.getInstance().start("vfw://0", targets);
            mnuItemStartWebcam.setText("Stop My Webcam");
         } else {
```

```
            com.jchat.utils.SwingUtilities.showErrorMessage("Webcam is already
up and running.");
         }
      } else {
         JChatVideoTransmitter.getInstance().stop();
         mnuItemStartWebcam.setText("Start My Webcam");
      }
   }

   private void
mnuItemStartAudioStreamActionPerformed(java.awt.event.ActionEvent evt) {
   if (mnuItemStartAudioStream.getText().startsWith("Start")) {
         if (!JChatAudioTransmitter.getInstance().isRunning()) {
            Vector targets = new Vector();
            Target aTraget = new Target("2224",
JChatClient.getMyProfile().getIpAddress(), "4442");
            targets.add(aTraget);
            JChatAudioTransmitter.getInstance().start("dsound://", targets);
            mnuItemStartAudioStream.setText("Stop Audio Stream");
         } else {
            com.jchat.utils.SwingUtilities.showErrorMessage("Audio Stream is
already up and running.");
         }
      } else {
         JChatAudioTransmitter.getInstance().stop();
         mnuItemStartAudioStream.setText("Start My Webcam");
      }
}

   private void mnuItemAboutActionPerformed(java.awt.event.ActionEvent evt) {
      AboutJDialog.setVisible(true);
   }
   /**
    * @param args the command line arguments
    */
   // Variables declaration - do not modify
   private javax.swing.JButton btnSendChat;
   private javax.swing.JList lstFriendsList;
   private javax.swing.JMenuBar mnuBar;
   private javax.swing.JMenu mnuFile;
   private javax.swing.JMenu mnuHelp;
   private javax.swing.JMenuItem mnuItemAbout;
   private javax.swing.JMenuItem mnuItemExit;
```

```
    private javax.swing.JMenuItem mnuItemStartAudioStream;
    private javax.swing.JMenuItem mnuItemStartWebcam;
    private javax.swing.JMenu mnuTools;
    private javax.swing.JPanel pnlChatInput;
    private javax.swing.JPanel pnlControls;
    private javax.swing.JPanel pnlMain;
    private javax.swing.JScrollPane scrollPaneChatWindow;
    private javax.swing.JScrollPane scrollPaneFriendsList;
    private javax.swing.JScrollPane scrollPaneInput;
    private javax.swing.JSplitPane splitPane;
    private javax.swing.JTextPane txtChatInput;
    private javax.swing.JTextArea txtChatWindow;
    // End of variables declaration
    private DefaultListModel listModel = new DefaultListModel();
    private About AboutJDialog = new About(this,true);
}
```

<**Τέλος κώδικα**>

Εδώ είναι το κεντρικό JFrame όπου υπάρχει η λίστα των χρηστών, μέσω του JButton μπορούμε να στείλουμε το μήνυμα που θέλουμε και να το δουν όλοι, μέσω του sendChat από το JChatClientInterface. Ακόμη μπορούμε μέσω της JmenuBar->Tools, να ενεργοποιήσουμε είτε το Video Chat είτε το Audio είτε και τα δυο, η Ανάλυση τους θα γίνει τελευταία για ευνόητους λόγους. Αν επιλέξουμε ένα χρήστη αυτόματα ανοίγει το PrivateChat μέσω της χρήσης της lstFriendslistValueChanged από την οποία παίρνουμε μέσω της ClientProfile το όνομα του χρήστη και βλέπουμε αν έχουμε ήδη ανοικτό παράθυρο μέσω της Busy οπότε ανοίγουμε ένα καινούργιο παράθυρο που πλέον είναι το Private (frmPrivateChat).

<**Αρχή κώδικα frmPrivateChat**>

```
package com.jchat.gui;

import com.jchat.Constants;
import com.jchat.JChatClient;
import com.jchat.client.ClientProfile;
import com.jchat.test.trasmitter.JChatAudioTransmitter;
import com.jchat.test.trasmitter.JChatVideoTransmitter;
import java.awt.BorderLayout;


public class FrmPrivateChat extends javax.swing.JFrame {
```

```
    /** Creates new form FrmPrivateChat */
    private ClientProfile chatterProfile = null;
    private PnlJChatStreaming pnlStreaming = null;

    public FrmPrivateChat(ClientProfile chatterProfile) {
        this.chatterProfile = chatterProfile;
        initComponents();
        loadData();
        loadStreaming();
        pack();
    }

    public void updateChatWindow(String text) {
        txtChatWindow.append(text + "\n");
        txtChat.setText("");
        try{
            txtChatWindow.setCaretPosition(txtChatWindow.getText().length() - 1);
        } catch (Exception ex){
        }
        this.requestFocus();
    }

    private void loadData() {
        if (chatterProfile != null) {
            lblChatterOther.setText("He/She is : " + chatterProfile.getNickname());
        }
        lblChatterMe.setText("I am  : " +
JChatClient.getMyProfile().getNickname());
        JChatClient.getMyProfile().setIsBusy(true);
        chatterProfile.setIsBusy(true);
    }

    public ClientProfile getChatterProfile() {
        return chatterProfile;
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
```

```
    private void initComponents() {
        java.awt.GridBagConstraints gridBagConstraints;

        pnlIntro = new javax.swing.JPanel();
        lblChatterMe = new javax.swing.JLabel();
        lblChatterOther = new javax.swing.JLabel();
        pnlMain = new javax.swing.JPanel();
        pnlChatInput = new javax.swing.JPanel();
        scrollPaneChatInput = new javax.swing.JScrollPane();
        txtChat = new javax.swing.JTextArea();
        pnlChatInputControls = new javax.swing.JPanel();
        btnSendChat = new javax.swing.JButton();
        pnlChatWindow = new javax.swing.JPanel();
        scrollPaneChatWindow = new javax.swing.JScrollPane();
        txtChatWindow = new javax.swing.JTextArea();

        setTitle("Private Chat");
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                onPrivateWindowCloseHandler(evt);
            }
        });


pnlIntro.setBorder(javax.swing.BorderFactory.createTitledBorder("Participants"));
        pnlIntro.setLayout(new java.awt.GridBagLayout());

        lblChatterMe.setFont(new java.awt.Font("Batang", 1, 12));
        lblChatterMe.setForeground(new java.awt.Color(51, 51, 255));
        lblChatterMe.setIcon(new javax.swing.ImageIcon("C:\\My
Drive\\Work\\RAC\\JChat\\resources\\user32.png")); // NOI18N
        lblChatterMe.setText("Chatter Me");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
        gridBagConstraints.weightx = 1.0;
        gridBagConstraints.insets = new java.awt.Insets(10, 10, 10, 10);
        pnlIntro.add(lblChatterMe, gridBagConstraints);

        lblChatterOther.setFont(new java.awt.Font("Batang", 1, 12));
        lblChatterOther.setForeground(new java.awt.Color(0, 153, 51));
        lblChatterOther.setIcon(new javax.swing.ImageIcon("C:\\My
Drive\\Work\\RAC\\JChat\\resources\\userb32.png")); // NOI18N
```

```
        lblChatterOther.setText("Chatter 2");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridy = 1;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
        gridBagConstraints.insets = new java.awt.Insets(0, 10, 10, 10);
        pnlIntro.add(lblChatterOther, gridBagConstraints);

        getContentPane().add(pnlIntro, java.awt.BorderLayout.NORTH);

        pnlMain.setLayout(new java.awt.BorderLayout());

        pnlChatInput.setBorder(javax.swing.BorderFactory.createTitledBorder("Chat
Input"));
        pnlChatInput.setLayout(new java.awt.BorderLayout());

        txtChat.setColumns(20);
        txtChat.setRows(5);
        scrollPaneChatInput.setViewportView(txtChat);

        pnlChatInput.add(scrollPaneChatInput, java.awt.BorderLayout.CENTER);

        pnlChatInputControls.setLayout(new java.awt.GridBagLayout());


btnSendChat.setText("<html><body><H4>Send<br/>Chat</H4></body></html>
");
        btnSendChat.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                btnSendChatActionPerformed(evt);
            }
        });
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.insets = new java.awt.Insets(10, 10, 10, 10);
        pnlChatInputControls.add(btnSendChat, gridBagConstraints);

        pnlChatInput.add(pnlChatInputControls, java.awt.BorderLayout.EAST);

        pnlMain.add(pnlChatInput, java.awt.BorderLayout.SOUTH);


pnlChatWindow.setBorder(javax.swing.BorderFactory.createTitledBorder("Chat
Details"));
```

```
        pnlChatWindow.setPreferredSize(new java.awt.Dimension(300, 200));
        pnlChatWindow.setLayout(new java.awt.BorderLayout());

        txtChatWindow.setColumns(20);
        txtChatWindow.setEditable(false);
        txtChatWindow.setRows(5);
        scrollPaneChatWindow.setViewportView(txtChatWindow);

        pnlChatWindow.add(scrollPaneChatWindow,
java.awt.BorderLayout.CENTER);

        pnlMain.add(pnlChatWindow, java.awt.BorderLayout.CENTER);

        getContentPane().add(pnlMain, java.awt.BorderLayout.CENTER);
    }// </editor-fold>
    private void loadStreaming() {
        try {

JChatVideoTransmitter.getInstance().addTarget(chatterProfile.getIpAddress(),
"6666");

JChatAudioTransmitter.getInstance().addTarget(chatterProfile.getIpAddress(),
"8888");

            pnlStreaming = new PnlJChatStreaming(chatterProfile, this);
            pnlMain.add(pnlStreaming, BorderLayout.EAST);
            pack();
        } catch (Exception ex) {
            ex.printStackTrace();
            com.jchat.utils.SwingUtilities.showErrorMessage(ex.getMessage());
        }
    }

    private void onPrivateWindowCloseHandler(java.awt.event.WindowEvent evt)
{
        try {
            try{
                pnlStreaming.close();

            } catch (Exception ex){
                ex.printStackTrace();
            }
            FrmJChatClient.getPrivateChatWindows().remove(this);
```

```
            JChatClient.getMyProfile().setIsBusy(false);

JChatVideoTransmitter.getInstance().removeTarget(chatterProfile.getIpAddress(),
"6666");
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    private void btnSendChatActionPerformed(java.awt.event.ActionEvent evt) {
        String chatMessage = Constants.CLIENT_PRIVATE_MESSAGE + " ";
        chatMessage +=
                chatterProfile.getNickname() + ";" +
                JChatClient.getMyProfile().getNickname() +
                ";" + txtChat.getText();
        JChatClient.getChatFrame().getChatClient().sendChat(chatMessage);
        updateChatWindow(JChatClient.getMyProfile().getNickname() + " says > "
+ txtChat.getText());
}
    // Variables declaration - do not modify
    private javax.swing.JButton btnSendChat;
    private javax.swing.JLabel lblChatterMe;
    private javax.swing.JLabel lblChatterOther;
    private javax.swing.JPanel pnlChatInput;
    private javax.swing.JPanel pnlChatInputControls;
    private javax.swing.JPanel pnlChatWindow;
    private javax.swing.JPanel pnlIntro;
    private javax.swing.JPanel pnlMain;
    private javax.swing.JScrollPane scrollPaneChatInput;
    private javax.swing.JScrollPane scrollPaneChatWindow;
    private javax.swing.JTextArea txtChat;
    private javax.swing.JTextArea txtChatWindow;
    // End of variables declaration
}
```

<**Τέλος Κώδικα**>

Μόλις καλέσουμε τη frmPrivateChat από τη frmJChatClient αυτομάτως διαβάζεται το Profile αυτού που μιλάμε μαζί δημιουργείτε το γραφικό περιβάλλον προσθέτουμε στο περιβάλλον το όνομα του χρήστη που μιλάμε και γίνεται Busy=true ούτως ώστε να ξέρουμε ότι υπάρχει ανοικτό PrivateChat με αυτόν το χρήστη, και ξεκινάμε το Streaming αναλόγως με το τι έχει ενεργοποιήσει ο καθένας, το οποίο θα αναλυθεί αργότερα.

<**Αρχη κωδικα About**>

```
package com.jchat.gui;


public class About extends javax.swing.JDialog {

    /** Creates new form About */
    public About(java.awt.Frame parent, boolean modal) {
        super(parent, modal);
        initComponents();
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jTextField1 = new javax.swing.JTextField();


setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE
);
        setTitle("About");
        setModal(true);
        setResizable(false);
        getContentPane().setLayout(new java.awt.FlowLayout());

        jTextField1.setEditable(false);
        jTextField1.setText("JChat Version 1.0 By Alexandros Kastanidis");
        getContentPane().add(jTextField1);

        pack();
    }// </editor-fold>


    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                About dialog = new About(new javax.swing.JFrame(), true);
```

```
            dialog.addWindowListener(new java.awt.event.WindowAdapter() {
                public void windowClosing(java.awt.event.WindowEvent e) {
                    System.exit(0);
                }
            });
            dialog.setVisible(true);
        }
    });
}

// Variables declaration - do not modify
private javax.swing.JTextField jTextField1;
// End of variables declaration

}
```

<**Τελος Κωδικα**>

Εδώ εχουμε ένα απλο JDialog που μας εμφανιζει ένα μικρο μηνυμα στο οσο αφορα το About της εργασιας.

Περαιτέρω Ανάλυση Client

Εδώ θα αναλύσουμε λίγο περισσότερο τον Client ,την χρήση κλάσεων και μεθόδων και το πώς καλούν η μια την άλλη.

Αφού τρέξουμε τη JChatClient και με τη βοήθεια της βοηθητικής IPAddressUlitily πάρουμε την IP καλούμε την JChatClientInterface να περάσουμε τα ορίσματα των Stream ,δημιουργούμε το frmJChatClient με όρισμα το JChatClientInterface που ήδη έχουμε περάσει τα ορίσματα των Stream διαβάζουμε το nickname και ξεκινάμε τη run όπου διαβάζει μηνύματα και τα αναλύει με τη χρήση της proccessMessage.
Αφού πλέον έχουμε το JFrame του client μας εκεί έχουμε κάποιες επιλογές όπως να μιλήσουμε στο κεντρικό JTextArea μέσω του JButton το οποίο στέλνει τα μηνύματα μέσω της sendChat της JChatClientInterface, ακόμη μπορούμε να ενεργοποιήσουμε το Video και/ή Audio μέσω της JMenuBar->Tools (θα επεξηγήσουμε την ακριβή Ανάλυση αργότερα αναλυτικά) είτε να επιλέξουμε ένα χρήστη και μέσω της lstFriendsListValueChanged να ανοίξουμε το frmPrivateChat και να μπορούμε να μιλήσουμε ή/και να ακούσουμε ή/και να δούμε τον άλλο χρήστη του οποίου το frmPrivateChat δεν θα ανοίξει αν δεν στείλουμε ένα κείμενο και με τη χρήση της deliverPrivateMessage να ανοίξει το παράθυρο σε αυτόν και να πάει το μήνυμα μας.

Αυτή είναι η σειρά την οποία καλεί η μια κλάση την άλλη και οι πλέον απαραίτητες ενέργειες και μέθοδοι που χρειάζονται για τη λειτουργία.

Κώδικας και Ανάλυση βοηθητικών μεθόδων :

<**Αρχή Κώδικα IPAddressUtility**>

```
package com.jchat.utils;


import java.io.*;
import java.net.*;
import java.util.*;
import static java.lang.System.out;

public class IPAddressUtility
{
   public static String getLocalIPAddress() throws SocketException {
      String ipAddress = null;
      Enumeration<NetworkInterface> nets =
NetworkInterface.getNetworkInterfaces();
      for (NetworkInterface netint : Collections.list(nets)){
         ipAddress = getLocalIPAddress(netint);
         if(ipAddress != null){
            return ipAddress;
         }
      }
      return null;
   }

   private static String getLocalIPAddress(NetworkInterface netint) throws
SocketException {
      Enumeration<InetAddress> inetAddresses = netint.getInetAddresses();
      for (InetAddress inetAddress : Collections.list(inetAddresses)) {
         if(inetAddress != null){
            if(!inetAddress.getHostAddress().startsWith("127") && inetAddress
instanceof Inet4Address){
               System.out.println("Returning:" + inetAddress.getHostAddress());
               return inetAddress.getHostAddress();
            }
```

```
            }
        }
        return null;
    }
}
```

<**Τέλος Κώδικα**>

Χρησιμοποιούμε αυτές τις μεθόδους για να πάρουμε την IP μέσω της διεύθυνσης του συγκεκριμένου Network Interface που Χρησιμοποιούμε.

<**Αρχή Κώδικα SwingUtilities**>

```
package com.jchat.utils;

import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class SwingUtilities {

   public static void centerFrame(JFrame component) {
    java.awt.Dimension screenSize =
java.awt.Toolkit.getDefaultToolkit().getScreenSize();
    java.awt.Dimension frameSize = component.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    component.setLocation((screenSize.width - frameSize.width) / 2,
(screenSize.height - frameSize.height) / 2);
  }

  public static void centerFrame(javax.swing.JInternalFrame internalFrame,
javax.swing.JFrame jFrame) {
    internalFrame.setVisible(false);
    internalFrame.setLocation((jFrame.getWidth() - internalFrame.getWidth()) / 2,
(jFrame.getHeight() - internalFrame.getHeight()) / 2 - 21);
    internalFrame.setVisible(true);
  }
```

```
    public static void showErrorMessage(String errMsg) {
    JOptionPane.showMessageDialog(null, errMsg, "Error Occured !!",
JOptionPane.ERROR_MESSAGE);
  }

  public static int showWarnMessage(String warnMsg) {
    return JOptionPane.showConfirmDialog(null, warnMsg, "Attention !!",
JOptionPane.YES_NO_OPTION, JOptionPane.WARNING_MESSAGE);
  }
}
```

<**Τέλος Κώδικα**>

Εδώ περιέχονται κάποιες μέθοδοι για το κεντράρισμα των JFrame και, για τους διαλόγους που χρησιμοποιούμε για διάφορα λάθη και σημειώσεις σε runtime.

<**Αρχή Κώδικα Constants**>

```
package com.jchat;

public class Constants {
    public static final String APP_TITLE = "JChat Client ~ (v 1.0)";

    public static final String SERVER_MESSAGE = "[ServerMessage]";
    public static final String SERVER_MESSAGE_JOIN_NOTIFICATION =
"[ServerMessage-Join]";
    public static final String SERVER_MESSAGE_LEFT_NOTIFICATION =
"[ServerMessage-Left]";
    public static final String SERVER_MESSAGE_CLIENT_LIST =
"[ServerMessage-Client-List]";

    public static final String CLIENT_MESSAGE = "[ClientMessage]";
    public static final String CLIENT_MESSAGE_JOIN = "[ClientMessage-Join]";
    public static final String CLIENT_MESSAGE_LEFT = "[ClientMessage-
Left]";

    public static final String CLIENT_PRIVATE_MESSAGE =
"[ClientPrivateMessage]";

    public static final String CLIENT_PRIVATE_CHAT_REQUEST =
"[ClientPrivateChatRequest]";
    public static final String CLIENT_PRIVATE_CHAT_RESPONSE_OK =
"[ClientPrivateChatReponseOK]";
```

```
   public static final String CLIENT_PRIVATE_CHAT_RESPONSE_DENY =
"[ClientPrivateChatReponseDeny]";
   public static final String CLIENT_PRIVATE_CHAT_RESPONSE_BUSY =
"[ClientPrivateChatReponseBusy]";
}
```

<**Τέλος Κώδικα**>

Εδώ υπάρχουν κάποια String τα οποία χρησιμοποιούμε για την επικοινωνία του προγράμματος μέσω της proccessMessage.

Ανάλυση και Κώδικας Panel για Audio/Video

<**Αρχή κώδικα PnlJChatStreaming**>

```
package com.jchat.gui;

import com.jchat.JChatClient;
import com.jchat.client.ClientProfile;
import com.jchat.test.receiver.JChatAudioReceiver;
import com.jchat.test.receiver.JChatVideoReceiver;
import com.jchat.test.receiver.Target;
import java.awt.BorderLayout;
import java.util.Vector;

public class PnlJChatStreaming extends javax.swing.JPanel {

   /** Creates new form PnlJChatStreaming */
   private ClientProfile otherParticipant = null;

   private JChatVideoReceiver selfVideoStreaming = null;

   private JChatVideoReceiver othersVideoStreaming = null;
   private JChatAudioReceiver othersAudioStreaming = null;


   private PnlJChatStreamPlayer selfStreamPlayer = null;
   private PnlJChatStreamPlayer otherStreamPlayer = null;

   private FrmPrivateChat privateChatWindow = null;
```

```
    public PnlJChatStreaming(ClientProfile otherParticipant, FrmPrivateChat
privateChatWindow) {
        this.otherParticipant = otherParticipant;
        this.privateChatWindow = privateChatWindow;
        initComponents();
        initStreaming();
    }

    public void close(){
        try{
            selfVideoStreaming.close();
        } catch (Exception ex){
            ex.printStackTrace();
        }
        try{
            othersVideoStreaming.close();
            othersAudioStreaming.close();
        } catch (Exception ex){
            ex.printStackTrace();
        }
    }

    public void updateWindow(){
        this.revalidate();
        privateChatWindow.pack();
    }

    private void initStreaming(){
        Target selfTarget = new Target("4444",
JChatClient.getMyProfile().getIpAddress(), "2222");


        Vector selfTargets = new Vector();
        selfTargets.add(selfTarget);


        selfVideoStreaming = new JChatVideoReceiver(selfTargets);

        Target otherVideoTarget = new Target("6666",
otherParticipant.getIpAddress(), "2222");
        Target otherAudioTarget = new Target("8888",
otherParticipant.getIpAddress(), "2224");
```

```
        Vector otherTargets = new Vector();
        otherTargets.add(otherVideoTarget);

        Vector otherAudioTargets = new Vector();
        otherAudioTargets.add(otherAudioTarget);

        othersVideoStreaming = new JChatVideoReceiver(otherTargets);
        othersAudioStreaming = new JChatAudioReceiver(otherAudioTargets);

        selfStreamPlayer = new
PnlJChatStreamPlayer(JChatClient.getMyProfile().getNickname(), this, true);
        otherStreamPlayer = new
PnlJChatStreamPlayer(otherParticipant.getNickname(), this, false);

        selfVideoStreaming.setStreamPlayer(selfStreamPlayer);
        othersVideoStreaming.setStreamPlayer(otherStreamPlayer);

        pnlMain.add(selfStreamPlayer, BorderLayout.NORTH);
        pnlMain.add(otherStreamPlayer, BorderLayout.SOUTH);

        otherStreamPlayer.setAudioStream(othersAudioStreaming);
        try{
           selfVideoStreaming.initialize();
        } catch (Exception ex){
           ex.printStackTrace();
           com.jchat.utils.SwingUtilities.showErrorMessage(ex.getMessage());
        }
        try{
           othersVideoStreaming.initialize();
           othersAudioStreaming.initialize();
        } catch (Exception ex){
           ex.printStackTrace();
           com.jchat.utils.SwingUtilities.showErrorMessage(ex.getMessage());
        }
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
```

```
        pnlMain = new javax.swing.JPanel();

        setLayout(new java.awt.BorderLayout());

        pnlMain.setLayout(new java.awt.BorderLayout());
        add(pnlMain, java.awt.BorderLayout.CENTER);
    }// </editor-fold>


    // Variables declaration - do not modify
    private javax.swing.JPanel pnlMain;
    // End of variables declaration

}
```

<**Τέλος κώδικα**>

<**Αρχή κώδικα PnlJChatStreamPlayer**>

```
package com.jchat.gui;

import com.jchat.test.receiver.JChatAudioReceiver;
import com.jchat.test.receiver.JChatVideoReceiver;
import com.jchat.test.receiver.JChatVideoReceiver.PlayerWindow;
import java.awt.BorderLayout;
import java.awt.CardLayout;
import java.awt.Component;
import javax.swing.DefaultListModel;

public class PnlJChatStreamPlayer extends javax.swing.JPanel {

    /** Creates new form PnlJChatStreamPlayer */

    private String title = "Stream Player";
    private PnlJChatStreaming streamingPanel = null;
    private JChatVideoReceiver videoStreamReceiver = null;
    private JChatAudioReceiver audioStreamReceiver = null;

    public PnlJChatStreamPlayer(String title, PnlJChatStreaming streamingPanel,
boolean onlyVideoStreaming) {
        this.title = title;
```

```
        this.streamingPanel = streamingPanel;
        initComponents();
        if (!onlyVideoStreaming) {
            cmbBufferType.addItem("Video");
            cmbBufferType.addItem("Audio");
        } else {
            cmbBufferType.addItem("Video");
        }
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        pnlMain = new javax.swing.JPanel();
        pnlIcon = new javax.swing.JPanel();
        lblIcon = new javax.swing.JLabel();
        pnlPlayer = new javax.swing.JPanel();
        pnlBufferControl = new javax.swing.JPanel();
        cmbBufferType = new javax.swing.JComboBox();
        lblCurrentLength = new javax.swing.JLabel();
        lblCurrentLengthValue = new javax.swing.JLabel();
        btnIncrease = new javax.swing.JButton();
        btnDecrease = new javax.swing.JButton();
        pnlControls = new javax.swing.JPanel();
        btnViewHide = new javax.swing.JButton();

        setBorder(javax.swing.BorderFactory.createTitledBorder(this.title));
        setLayout(new java.awt.BorderLayout());

        pnlMain.setLayout(new java.awt.CardLayout());

        pnlIcon.setLayout(new java.awt.BorderLayout());

        lblIcon.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        lblIcon.setIcon(new javax.swing.ImageIcon("C:\\My
Drive\\Work\\RAC\\JChat\\resources\\user.png")); // NOI18N
        pnlIcon.add(lblIcon, java.awt.BorderLayout.CENTER);
```

```
pnlMain.add(pnlIcon, "PANEL_ICON");

pnlPlayer.setLayout(new java.awt.BorderLayout());

pnlBufferControl.add(cmbBufferType);

lblCurrentLength.setText("Buffer Length:");
pnlBufferControl.add(lblCurrentLength);

lblCurrentLengthValue.setText("N/A");
pnlBufferControl.add(lblCurrentLengthValue);

btnIncrease.setText("+");
btnIncrease.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      btnIncreaseActionPerformed(evt);
   }
});
pnlBufferControl.add(btnIncrease);

btnDecrease.setText("-");
btnDecrease.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      btnDecreaseActionPerformed(evt);
   }
});
pnlBufferControl.add(btnDecrease);

pnlPlayer.add(pnlBufferControl, java.awt.BorderLayout.SOUTH);

pnlMain.add(pnlPlayer, "PANEL_PLAYER");

add(pnlMain, java.awt.BorderLayout.CENTER);

btnViewHide.setText("View Streaming");
btnViewHide.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      btnViewHideActionPerformed(evt);
   }
});
pnlControls.add(btnViewHide);

add(pnlControls, java.awt.BorderLayout.SOUTH);
```

```
    }// </editor-fold>
    private void btnViewHideActionPerformed(java.awt.event.ActionEvent evt) {
        if (btnViewHide.getText().contains("View")) {
            changeDisplay(PANEL_PLAYER);
        } else {
            changeDisplay(PANEL_ICON);
        }
    }

    private void btnIncreaseActionPerformed(java.awt.event.ActionEvent evt) {
        if (cmbBufferType.getSelectedIndex() == 0) {
            if (videoStreamReceiver != null) {
                try {

videoStreamReceiver.setBufferLength(videoStreamReceiver.getBufferLength() +
1);
                    updateBufferLength();
                } catch (Exception ex) {
                    com.jchat.utils.SwingUtilities.showErrorMessage("Error occured
while updating buffet size:\n" + ex.getMessage());
                }
            }
        } else {
            if (audioStreamReceiver != null) {
                try {

audioStreamReceiver.setBufferLength(audioStreamReceiver.getBufferLength() +
1);
                    updateBufferLength();
                } catch (Exception ex) {
                    com.jchat.utils.SwingUtilities.showErrorMessage("Error occured
while updating buffet size:\n" + ex.getMessage());
                }
            }
        }

    }

    private void btnDecreaseActionPerformed(java.awt.event.ActionEvent evt) {
        if (cmbBufferType.getSelectedIndex() == 0) {
            if (videoStreamReceiver != null) {
                try {
```

```
videoStreamReceiver.setBufferLength(videoStreamReceiver.getBufferLength() -
1);
                updateBufferLength();
            } catch (Exception ex) {
                com.jchat.utils.SwingUtilities.showErrorMessage("Error occured
while updating buffet size:\n" + ex.getMessage());
            }
         }
      } else {
         if (audioStreamReceiver != null) {
            try {

audioStreamReceiver.setBufferLength(audioStreamReceiver.getBufferLength() -
1);
                updateBufferLength();
            } catch (Exception ex) {
                com.jchat.utils.SwingUtilities.showErrorMessage("Error occured
while updating buffet size:\n" + ex.getMessage());
            }
         }
      }
   }

   public void playStream(JChatVideoReceiver streamReceiver) {
      this.videoStreamReceiver = streamReceiver;
      Component comp[] = pnlPlayer.getComponents();
      for (int i = 0; i < comp.length; i++) {
         if (comp[i] instanceof PlayerWindow) {
            pnlPlayer.remove(comp[i]);
            break;
         }
      }
      pnlPlayer.add(streamReceiver.getPlayerWindow(), BorderLayout.CENTER);
      pnlPlayer.revalidate();
      this.revalidate();
      updateBufferLength();
      streamingPanel.updateWindow();
      changeDisplay(PANEL_PLAYER);
   }

   public void setAudioStream(JChatAudioReceiver audioStreamReceiver) {
      this.audioStreamReceiver = audioStreamReceiver;
```

```
    }

    private void changeDisplay(String panel) {
        if (panel.contains(PANEL_ICON)) {
            btnViewHide.setText("View Streaming");
        } else {
            btnViewHide.setText("Hide Streaming");
        }
        CardLayout cl = (CardLayout) (pnlMain.getLayout());
        cl.show(pnlMain, panel);
    }
    // Variables declaration - do not modify
    private javax.swing.JButton btnDecrease;
    private javax.swing.JButton btnIncrease;
    private javax.swing.JButton btnViewHide;
    private javax.swing.JComboBox cmbBufferType;
    private javax.swing.JLabel lblCurrentLength;
    private javax.swing.JLabel lblCurrentLengthValue;
    private javax.swing.JLabel lblIcon;
    private javax.swing.JPanel pnlBufferControl;
    private javax.swing.JPanel pnlControls;
    private javax.swing.JPanel pnlIcon;
    private javax.swing.JPanel pnlMain;
    private javax.swing.JPanel pnlPlayer;
    // End of variables declaration
    private static String PANEL_ICON = "PANEL_ICON";
    private static String PANEL_PLAYER = "PANEL_PLAYER";

    private void updateBufferLength() {
        try {
            if (cmbBufferType.getSelectedIndex() == 0) {
                if (videoStreamReceiver.getBufferLength() == 1) {
                    lblCurrentLengthValue.setText("N/A");
                } else {

lblCurrentLengthValue.setText(String.valueOf(videoStreamReceiver.getBufferLe
ngth()));
                }
            } else {
                if (audioStreamReceiver.getBufferLength() == 1) {
                    lblCurrentLengthValue.setText("N/A");
                } else {
```

```
lblCurrentLengthValue.setText(String.valueOf(audioStreamReceiver.getBufferLe
ngth()));
                }

            }
        } catch (Exception ex) {
            lblCurrentLengthValue.setText("N/A");
        }
    }
}
```

<**Τέλος κώδικα**>

Με το άνοιγμα του private chat αφού δημιουργηθεί το περιβάλλον και γίνουν load τα διάφορα δεδομένα με τη χρήση της loadStreaming προστίθεται ο προορισμός για το audio και video transmit μέσω της addtarget των αντιστοίχων κλάσεων, και μετά δημιουργούμε το PnlJChatStreaming ούτως ώστε να δούμε εκεί μέσα τα streaming.
Με την κλήση αυτής αφού αρχικοποιήσουμε το προφίλ του άλλου χρήστη και το δικό μας FrmPrivateChat δημιουργείται το περιβάλλον και με την χρήση της initStreaming ξεκινάμε την λειτουργία η οποία γίνεται ως εξής:

Τοποθετούμε εμάς ως νέο χρήστη στους Target, και δημιουργούμε ένα νέο JChatVideoReceiver MONO με εμάς παραλήπτη (αφού δεν θέλουμε να ακούμε την φωνή μας), προσθέτουμε ένα Target για Video και ένα για Audio ξεχωριστά για τον άλλο χρήστη, και δημιουργούμε ένα JChatVideoReceiver και ένα JChatAudioReceiver για τον άλλο χρήστη για να έχουμε και φωνή και εικόνα, και δημιουργούμε τα PnlJChatStreamPlayer για εμάς με μόνο Video Buffer και ένα άλλο για τον άλλο χρήστη με Audio και Video (Αφού βέβαια έχει ενεργοποιήσει και τα 2), και τελικά προσθέτουμε τα 2 panel το δικό μας πάνω (NORTH) και του άλλου χρήστη κάτω (SOUTH). Αφού γίνουν όλα αυτά με τα initialize πηγαίνουμε στα αντίστοιχα JChatVideoReceiver και JChatAudioReceiver όπου και θα εξηγήσουμε παρακάτω την συνέχεια.

Ακόμη στην κλάση PnlJChatStreamPlayer τα Jbuttons για το ανέβασμα και κατέβασμα του Buffer, την playStream που χρησιμοποιείται σε περίπτωση κάποιας αλλαγής από την controllerUpdate η οποία λειτουργεί όταν γίνεται κάποιο Event, την close που απλά κλείνει τα streams, την changeDisplay που αλλάζει το τι βλέπουμε όταν πατάμε το Hide ή View Streaming ανάλογα, και η updateBufferLength η οποία δείχνει το μέγεθος του Buffer.

Ανάλυση και κώδικας Transmitter Ήχου και Βίντεο

<**Αρχή Κώδικα JChatVideoTransmitter** >

```
package com.jchat.test.trasmitter;

import java.awt.*;
import java.io.*;
import java.net.InetAddress;
import java.util.*;
import javax.media.*;
import javax.media.protocol.*;
import javax.media.format.*;
import javax.media.control.QualityControl;
import javax.media.control.TrackControl;
import javax.media.rtp.*;
import javax.media.rtp.event.*;
import javax.media.rtp.rtcp.*;

public class JChatVideoTransmitter implements ReceiveStreamListener,
RemoteListener,
      ControllerListener {

   private MediaLocator videoLocator;
   private Processor videoProcessor = null;
   private RTPManager videoRtpMgrs[];
   private int videoLocalPorts[];
   private DataSource videoDatasource = null;
   private int video_local_data_port;
   private static boolean isRunning = false;
   private static JChatVideoTransmitter transmitter = null;

   private JChatVideoTransmitter() {
      video_local_data_port = 2222;
   }

   public static JChatVideoTransmitter getInstance() {
      if (transmitter == null) {
         transmitter = new JChatVideoTransmitter();
      }
      return transmitter;
   }
```

```
public static boolean isRunning() {
   return isRunning;
}

public static void setRunning(boolean isRunning) {
   JChatVideoTransmitter.isRunning = isRunning;
}


public synchronized String start(String filename, Vector targets) {
   String result;
   setRunning(true);

   videoLocator = new MediaLocator(filename);
   result = createProcessor();
   if (result != null) {
      return result;
   }

   result = createVideoTransmitter(targets);

   if (result != null) {
      videoProcessor.close();
      videoProcessor = null;

      return result;
   }

   videoProcessor.start();
   return null;
}


private String createVideoTransmitter(Vector targets) {

   PushBufferDataSource pbds = (PushBufferDataSource) videoDatasource;
   PushBufferStream pbss[] = pbds.getStreams();

   videoRtpMgrs = new RTPManager[pbss.length];
   videoLocalPorts = new int[pbss.length];

   SessionAddress localAddr;
```

```
        SendStream sendStream;
        int port;

        for (int i = 0; i < pbss.length; i++) {
            try {
                videoRtpMgrs[i] = RTPManager.newInstance();

                port = video_local_data_port + 2 * i;

                videoLocalPorts[i] = port;

                localAddr = new SessionAddress(InetAddress.getLocalHost(),
                        port);

                videoRtpMgrs[i].initialize(localAddr);
                videoRtpMgrs[i].addReceiveStreamListener(this);
                videoRtpMgrs[i].addRemoteListener(this);

                for (int k = 0; k < targets.size(); k++) {
                    Target target = (Target) targets.elementAt(k);

                    int targetPort = new Integer(target.port).intValue();

                    addTarget(videoLocalPorts[i], videoRtpMgrs[i], target.ip, targetPort +
2 * i);
                }
                sendStream = videoRtpMgrs[i].createSendStream(videoDatasource, i);
                sendStream.start();
            } catch (Exception e) {
                e.printStackTrace();

                return e.getMessage();
            }
        }

        return null;
    }

    public void addTarget(String ip, String videoPort) {
        try {
            if (ip.startsWith("/")) {
                ip = ip.substring(1);
            }
```

```
            int targetPort = new Integer(videoPort).intValue();
            addTarget(videoLocalPorts[0], videoRtpMgrs[0], ip, targetPort);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    public void addTarget(int localPort, RTPManager mgr, String ip, int port) {
        try {
            SessionAddress addr = new SessionAddress(InetAddress.getByName(ip),
                    new Integer(port).intValue());
            mgr.addTarget(addr);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void removeTarget(String ip, String port) {
        try {
            if (ip.trim().startsWith("/")) {
                ip = ip.substring(1);
            }
            SessionAddress addr = new SessionAddress(InetAddress.getByName(ip),
                    new Integer(port).intValue());
            for (int i = 0; i < videoRtpMgrs.length; i++) {
                videoRtpMgrs[i].removeTarget(addr, "target removed from
transmitter.");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    boolean looping = true;

    public void controllerUpdate(ControllerEvent ce) {
        System.out.println(ce);
        if (ce instanceof DurationUpdateEvent) {
            Time duration = ((DurationUpdateEvent) ce).getDuration();

            System.out.println("duration: " + duration.getSeconds());
        } else if (ce instanceof EndOfMediaEvent) {
            System.out.println("END OF MEDIA - looping=" + looping);
```

```
        if (looping) {
            videoProcessor.setMediaTime(new Time(0));
            videoProcessor.start();
        }
    }
}

public void setLooping(boolean flag) {
    looping = flag;
}

public void update(ReceiveStreamEvent event) {
    String timestamp = getTimestamp();

    StringBuffer sb = new StringBuffer();

    if (event instanceof InactiveReceiveStreamEvent) {
        sb.append(timestamp + " Inactive Receive Stream");
    } else if (event instanceof ByeEvent) {
        sb.append(timestamp + " Bye");
    } else {
        System.out.println("ReceiveStreamEvent: " + event);
    }
}

public void update(RemoteEvent event) {
    String timestamp = getTimestamp();

    if (event instanceof ReceiverReportEvent) {
        ReceiverReport rr = ((ReceiverReportEvent) event).getReport();

        StringBuffer sb = new StringBuffer();

        sb.append(timestamp + " RR");

        if (rr != null) {
            Participant participant = rr.getParticipant();

            if (participant != null) {
                sb.append(" from " + participant.getCNAME());
                sb.append(" ssrc=" + rr.getSSRC());
            } else {
                sb.append(" ssrc=" + rr.getSSRC());
```

```
            }
        }
    } else {
        System.out.println("RemoteEvent: " + event);
    }
}

private String getTimestamp() {
    String timestamp;

    Calendar calendar = Calendar.getInstance();

    int hour = calendar.get(Calendar.HOUR_OF_DAY);

    String hourStr = formatTime(hour);

    int minute = calendar.get(Calendar.MINUTE);

    String minuteStr = formatTime(minute);

    int second = calendar.get(Calendar.SECOND);

    String secondStr = formatTime(second);

    timestamp = hourStr + ":" + minuteStr + ":" + secondStr;

    return timestamp;
}

private String formatTime(int time) {
    String timeStr;

    if (time < 10) {
        timeStr = "0" + time;
    } else {
        timeStr = "" + time;
    }

    return timeStr;
}


public void stop() {
```

```
    synchronized (this) {
        setRunning(false);
        if (videoProcessor != null) {
            videoProcessor.stop();
            videoProcessor.close();
            videoProcessor = null;
            if (videoDatasource != null) {
                videoDatasource.disconnect();
            }
            for (int i = 0; i < videoRtpMgrs.length; i++) {
                videoRtpMgrs[i].removeTargets("Session ended.");
                videoRtpMgrs[i].dispose();
            }
        }
    }
}

public String createProcessor() {
    if (videoLocator == null) {
        return "Locator is null";
    }

    try {
        videoProcessor = javax.media.Manager.createProcessor(videoLocator);
        videoProcessor.addControllerListener(this);
    } catch (NoProcessorException npe) {
        npe.printStackTrace();
        return "Couldn't create processor";
    } catch (IOException ioe) {
        ioe.printStackTrace();
        return "IOException creating processor";
    }

    boolean result = waitForState(videoProcessor, Processor.Configured);

    if (result == false) {
        return "Couldn't configure video processor";
    }




    TrackControl[] tracks = videoProcessor.getTrackControls();
```

```
        if (tracks == null || tracks.length < 1) {
            return "Couldn't find tracks in processor";
        }

        .
        ContentDescriptor cd = new
ContentDescriptor(ContentDescriptor.RAW_RTP);

        videoProcessor.setContentDescriptor(cd);

        Format supported[];
        Format chosen;
        boolean atLeastOneTrack = false;


        for (int i = 0; i < tracks.length; i++) {
            Format format = tracks[i].getFormat();
            if (tracks[i].isEnabled()) {

                supported = tracks[i].getSupportedFormats();



                if (supported.length > 0) {
                    if (supported[0] instanceof VideoFormat) {
                        chosen = checkForVideoSizes(tracks[i].getFormat(),
                                supported[0]);
                    } else {
                        chosen = supported[0];
                    }
                    tracks[i].setFormat(chosen);
                    System.err.println("Track " + i + " is set to transmit as:");
                    System.err.println("  " + chosen);
                    atLeastOneTrack = true;
                } else {
                    tracks[i].setEnabled(false);
                }
            } else {
                tracks[i].setEnabled(false);
            }
        }
```

```
    if (!atLeastOneTrack) {
        return "Couldn't set any of the tracks to a valid RTP format";
    }


    result = waitForState(videoProcessor, Controller.Realized);
    if (result == false) {
        return "Couldn't video realize processor";
    }

    setJPEGQuality(videoProcessor, 0.5f);

    videoDatasource = videoProcessor.getDataOutput();

    if (videoDatasource == null) {
        System.out.println("*** video Dataoutput is null");
    }
    return null;
}
static SessionAddress destAddr1,  destAddr2;


Format checkForVideoSizes(Format original, Format supported) {

    int width, height;
    Dimension size = ((VideoFormat) original).getSize();
    Format jpegFmt = new Format(VideoFormat.JPEG_RTP);
    Format h263Fmt = new Format(VideoFormat.H263_RTP);

    if (supported.matches(jpegFmt)) {
        width = (size.width % 8 == 0 ? size.width : (int) (size.width / 8) * 8);
        height = (size.height % 8 == 0 ? size.height : (int) (size.height / 8) * 8);
    } else if (supported.matches(h263Fmt)) {
        if (size.width < 128) {
            width = 128;
            height = 96;
        } else if (size.width < 176) {
            width = 176;
            height = 144;
        } else {
            width = 352;
            height = 288;
```

```
        }
    } else {
        return supported;
    }

    return (new VideoFormat(null,
            new Dimension(width, height),
            Format.NOT_SPECIFIED,
            null,
            Format.NOT_SPECIFIED)).intersects(supported);
}


void setJPEGQuality(Player p, float val) {

    Control cs[] = p.getControls();
    QualityControl qc = null;
    VideoFormat jpegFmt = new VideoFormat(VideoFormat.JPEG);

    for (int i = 0; i < cs.length; i++) {

        if (cs[i] instanceof QualityControl &&
                cs[i] instanceof Owned) {
            Object owner = ((Owned) cs[i]).getOwner();

            if (owner instanceof Codec) {
                Format fmts[] = ((Codec) owner).getSupportedOutputFormats(null);
                for (int j = 0; j < fmts.length; j++) {
                    if (fmts[j].matches(jpegFmt)) {
                        qc = (QualityControl) cs[i];
                        qc.setQuality(val);
                        System.err.println("- Setting quality to " +
                                val + " on " + qc);
                        break;
                    }
                }
            }
            if (qc != null) {
                break;
            }
        }
    }
}
```

```
private Integer stateLock = new Integer(0);
private boolean failed = false;

Integer getStateLock() {
   return stateLock;
}

void setFailed() {
   failed = true;
}

private synchronized boolean waitForState(Processor p, int state) {
   p.addControllerListener(new StateListener());
   failed = false;

   if (state == Processor.Configured) {
      p.configure();
   } else if (state == Processor.Realized) {
      p.realize();
   }


   while (p.getState() < state && !failed) {
      synchronized (getStateLock()) {
         try {
            getStateLock().wait();
         } catch (InterruptedException ie) {
            return false;
         }
      }
   }

   if (failed) {
      return false;
   } else {
      return true;
   }
}

class StateListener implements ControllerListener {

   public void controllerUpdate(ControllerEvent ce) {
```

```
            if (ce instanceof ControllerClosedEvent) {
                setFailed();
            }

            if (ce instanceof ControllerEvent) {
                synchronized (getStateLock()) {
                    getStateLock().notifyAll();
                }
            }
        }
    }

    public static void main(String args[]) {

    }
}
```

<**Τέλος κώδικα**>

Με την χρήση μέσω της JMenubar->Tools->Start My Webcam ξεκινάμε το JChatVideoTransmitter ο οποίος ξεκινάει με την start και παίρνει όρισμα "vfw://0" και τους targets.
Με την χρήση της start δημιουργούμε ένα νέο MediaLocator ο οποίος χρησιμοποιείται μαζί με το όρισμα από vfw://0 και αυτόματα δημιουργείται ένας νέος Player με το $1^{ο}$ stream το οποίο αναγνωρίζεται στη συνεδρία. Από κάτω με τη χρήση τη createProcessor δημιουργούμε τον Processor μέσω του MediaLocator ούτως ώστε να διαβάζουμε τα RTP Streams και προσθέτουμε ένα listener για να μπορούμε να επεξεργαζόμαστε τα διάφορα events. Μέσω της waitForState μεταφέρουμε τον processor σε configured στάδιο. Μέσω της TrackControls παίρνουμε τα track των πολυμέσων (Video) και ελέγχουμε αν έχουμε έστω ένα για χρήση, και χρησιμοποιούμε το ContentDescriptor ώστε να επιλέξουμε το Format που θα χρησιμοποιήσουμε. Αφού κάνουμε και αυτό βρίσκουμε μέσω των Tracks τα υποστηριζόμενα tracks, μέσω της checkForVideoSizes ελέγχουμε τι είδους είναι αυτό το track (Jpeg, H263) και αφού επιλέξουμε με τη setFormat επιλεγούμε πλέον τι format θα χρησιμοποιήσουμε για το video (JPEG), πάλι μέσω της waitForState πηγαίνουμε τον processor σε Realized στάδιο, τοποθετούμε την συμπίεση του Jpeg (setJPEGQuality).Με την χρήση του getDataOutput περνούμε την έξοδο του processor ώστε να την χρησιμοποιήσουμε στον player μας.
Αφού πάρουμε τη έξοδο μπορούμε να τη χρησιμοποιήσουμε στον player μας, με την createVideoTransmitter, παίρνουμε μέσω της PushBufferDataSource το Datasource που πήραμε από το DataOutput του Processor και χρησιμοποιούμε το PushBufferStream για την χρήση του ως stream για το PushBufferDataSource.

Παρακάτω δημιουργούμε ένα νέο RTP session manager για κάθε stream το οποίο λαμβάνεται , διαβάζουμε την διεύθυνση και δίνουμε πόρτες με αυτόν τον τρόπο για να μην έχουμε conflicts σε πόρτες, κάνουμε initialize τον RTP session manager, καλούμε την addReceiveStreamListener για να μπορούμε αργότερα να χρησιμοποιήσουμε τον listener και την addRemoteListener για να μπορούμε να χειριστούμε τον listener απομακρυσμένα, καλούμε την createSendStream για να περάσουμε την DataSource και ένα δείκτη Stream και ξεκινάμε τον session manager με τη sendstream.start οπού γυρνάμε πίσω εκεί οπού ξεκινάει και ο Processor οπότε παίρνουμε το DataOutput μέσω αυτού το DataSource για να το πάρει ο RTP session manager και να έχουμε την μετάδοση.
Ακόμη έχουμε κάποιες συναρτήσεις όπως controllerupdate η οποία καλείται όποτε έχουμε ένα Event, 2 συναρτήσεις update αναλόγως και πάλι τα Events και διάφορες συναρτήσεις για την ευκολία μας όπως για την χρήση ώρας/ημερομηνίας, να δούμε σε τι state είναι ο Processor.

<**Αρχή κώδικα JChatAudioTransmitter**>

```
package com.jchat.test.trasmitter;

import java.awt.*;
import java.io.*;
import java.net.InetAddress;
import java.util.*;
import javax.media.*;
import javax.media.protocol.*;
import javax.media.format.*;
import javax.media.control.TrackControl;
import javax.media.rtp.*;
import javax.media.rtp.event.*;
import javax.media.rtp.rtcp.*;

public class JChatAudioTransmitter implements ReceiveStreamListener,
RemoteListener,
      ControllerListener {

   private MediaLocator audioLocator;
   private Processor audioProcessor = null;
   private RTPManager audioRtpMgrs[];
   private int audioLocalPorts[];
   private DataSource audioDatasource = null;
   private int audio_local_data_port;
   private static boolean isRunning = false;
```

```
private static JChatAudioTransmitter transmitter = null;

private JChatAudioTransmitter() {
   audio_local_data_port = 1111;
}

public static JChatAudioTransmitter getInstance() {
   if (transmitter == null) {
      transmitter = new JChatAudioTransmitter();
   }
   return transmitter;
}

public static boolean isRunning() {
   return isRunning;
}

public static void setRunning(boolean isRunning) {
   JChatAudioTransmitter.isRunning = isRunning;
}

public synchronized String start(String filename, Vector targets) {

   System.out.println("Running audio stream..");

   String result;
   setRunning(true);

   audioLocator = new MediaLocator(filename);
   result = createProcessor();
   if (result != null) {
      return result;
   }


   result = createAudioTransmitter(targets);

   if (result != null) {
      audioProcessor.close();
      audioProcessor = null;

      return result;
   }
```

```
        audioProcessor.start();
        return null;
    }


    private String createAudioTransmitter(Vector targets) {
        PushBufferDataSource pbds = (PushBufferDataSource) audioDatasource;
        PushBufferStream pbss[] = pbds.getStreams();

        audioRtpMgrs = new RTPManager[pbss.length];
        audioLocalPorts = new int[pbss.length];
        SessionAddress localAddr;

        SendStream sendStream;
        int port;

        for (int i = 0; i < pbss.length; i++) {
            try {
                audioRtpMgrs[i] = RTPManager.newInstance();

                port = audio_local_data_port + 2 * i;

                audioLocalPorts[i] = port;

                localAddr = new SessionAddress(InetAddress.getLocalHost(),
                        port);

                audioRtpMgrs[i].initialize(localAddr);
                audioRtpMgrs[i].addReceiveStreamListener(this);
                audioRtpMgrs[i].addRemoteListener(this);

                for (int k = 0; k < targets.size(); k++) {
                    Target target = (Target) targets.elementAt(k);

                    int targetPort = new Integer(target.port).intValue();

                    addTarget(audioLocalPorts[i], audioRtpMgrs[i], target.ip, targetPort +
2 * i);
                }

                sendStream = audioRtpMgrs[i].createSendStream(audioDatasource, i);
                sendStream.start();
            } catch (Exception e) {
```

```
            e.printStackTrace();

            return e.getMessage();
         }
      }

      return null;
   }

   public void addTarget(String ip, String videoPort) {
      try {
         if (ip.startsWith("/")) {
            ip = ip.substring(1);
         }
         int targetPort = new Integer(videoPort).intValue();
         addTarget(audioLocalPorts[0], audioRtpMgrs[0], ip, targetPort);
      } catch (Exception ex) {
         ex.printStackTrace();
      }
   }

   public void addTarget(int localPort, RTPManager mgr, String ip, int port) {
      try {
         SessionAddress addr = new SessionAddress(InetAddress.getByName(ip),
               new Integer(port).intValue());
         mgr.addTarget(addr);

      } catch (Exception e) {
         e.printStackTrace();
      }
   }


   public void removeTarget(String ip, String port) {
      try {
         if (ip.trim().startsWith("/")) {
            ip = ip.substring(1);
         }
         SessionAddress addr = new SessionAddress(InetAddress.getByName(ip),
               new Integer(port).intValue());
         for (int i = 0; i < audioRtpMgrs.length; i++) {
            audioRtpMgrs[i].removeTarget(addr, "target removed from
transmitter.");
```

```
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

boolean looping = true;


public void controllerUpdate(ControllerEvent ce) {
    System.out.println(ce);
    if (ce instanceof DurationUpdateEvent) {
        Time duration = ((DurationUpdateEvent) ce).getDuration();

        System.out.println("duration: " + duration.getSeconds());
    } else if (ce instanceof EndOfMediaEvent) {
        System.out.println("END OF MEDIA - looping=" + looping);
        if (looping) {
            audioProcessor.setMediaTime(new Time(0));
            audioProcessor.start();
        }
    }
}

public void setLooping(boolean flag) {
    looping = flag;
}


public void update(ReceiveStreamEvent event) {
    String timestamp = getTimestamp();

    StringBuffer sb = new StringBuffer();

    if (event instanceof InactiveReceiveStreamEvent) {
        sb.append(timestamp + " Inactive Receive Stream");
    } else if (event instanceof ByeEvent) {
        sb.append(timestamp + " Bye");
    } else {
        System.out.println("ReceiveStreamEvent: " + event);
    }
}
```

```
public void update(RemoteEvent event) {
   String timestamp = getTimestamp();

   if (event instanceof ReceiverReportEvent) {
      ReceiverReport rr = ((ReceiverReportEvent) event).getReport();

      StringBuffer sb = new StringBuffer();

      sb.append(timestamp + " RR");

      if (rr != null) {
         Participant participant = rr.getParticipant();

         if (participant != null) {
            sb.append(" from " + participant.getCNAME());
            sb.append(" ssrc=" + rr.getSSRC());
         } else {
            sb.append(" ssrc=" + rr.getSSRC());
         }
      }
   } else {
      System.out.println("RemoteEvent: " + event);
   }
}

private String getTimestamp() {
   String timestamp;

   Calendar calendar = Calendar.getInstance();

   int hour = calendar.get(Calendar.HOUR_OF_DAY);

   String hourStr = formatTime(hour);

   int minute = calendar.get(Calendar.MINUTE);

   String minuteStr = formatTime(minute);

   int second = calendar.get(Calendar.SECOND);

   String secondStr = formatTime(second);

   timestamp = hourStr + ":" + minuteStr + ":" + secondStr;
```

```
    return timestamp;
}

private String formatTime(int time) {
    String timeStr;

    if (time < 10) {
        timeStr = "0" + time;
    } else {
        timeStr = "" + time;
    }

    return timeStr;
}

public void stop() {
    synchronized (this) {
        setRunning(false);
        if (audioProcessor != null) {
            audioProcessor.stop();
            audioProcessor.close();
            audioProcessor = null;
            if (audioDatasource != null) {
                audioDatasource.disconnect();
            }
            for (int i = 0; i < audioRtpMgrs.length; i++) {
                audioRtpMgrs[i].removeTargets("Session ended.");
                audioRtpMgrs[i].dispose();
            }
        }
    }
}

public String createProcessor() {
    if (audioLocator == null) {
        return "Locator is null";
    }

    try {
        audioProcessor = javax.media.Manager.createProcessor(audioLocator);
        audioProcessor.addControllerListener(this);
    } catch (NoProcessorException npe) {
```

```
            npe.printStackTrace();
            return "Couldn't create processor";
        } catch (IOException ioe) {
            ioe.printStackTrace();
            return "IOException creating processor";
        }

        boolean result = waitForState(audioProcessor, Processor.Configured);

        if (result == false) {
            return "Couldn't configure audio processor";
        }

        TrackControl[] tracks = audioProcessor.getTrackControls();


        if (tracks == null || tracks.length < 1) {
            return "Couldn't find tracks in processor";
        }


        ContentDescriptor cd = new
ContentDescriptor(ContentDescriptor.RAW_RTP);

        audioProcessor.setContentDescriptor(cd);

        Format supported[];
        Format chosen;
        boolean atLeastOneTrack = false;

        for (int i = 0; i < tracks.length; i++) {
            Format format = tracks[i].getFormat();
            if (tracks[i].isEnabled()) {
                supported = tracks[i].getSupportedFormats();

                if (supported.length > 0) {

                    chosen = supported[0];

                    tracks[i].setFormat(chosen);
                    System.err.println("Track " + i + " is set to transmit as:");
                    System.err.println("  " + chosen);
                    atLeastOneTrack = true;
```

```
            } else {
                tracks[i].setEnabled(false);
            }
        } else {
            tracks[i].setEnabled(false);
        }
    }

    if (!atLeastOneTrack) {
        return "Couldn't set any of the tracks to a valid RTP format";
    }

   result = waitForState(audioProcessor, Controller.Realized);
    if (result == false) {
        return "Couldn't realize audio processor";
    }

    audioDatasource = audioProcessor.getDataOutput();

    if (audioDatasource == null) {
        System.out.println("*** audio Dataoutput is null");
    }

    return null;
}
static SessionAddress destAddr1,  destAddr2;

private Integer stateLock = new Integer(0);
private boolean failed = false;

Integer getStateLock() {
    return stateLock;
}

void setFailed() {
    failed = true;
}

private synchronized boolean waitForState(Processor p, int state) {
    p.addControllerListener(new StateListener());
    failed = false;

    if (state == Processor.Configured) {
```

```
        p.configure();
    } else if (state == Processor.Realized) {
        p.realize();
    }

    while (p.getState() < state && !failed) {
        synchronized (getStateLock()) {
            try {
                getStateLock().wait();
            } catch (InterruptedException ie) {
                return false;
            }
        }
    }

    if (failed) {
        return false;
    } else {
        return true;
    }
}


class StateListener implements ControllerListener {

    public void controllerUpdate(ControllerEvent ce) {

        if (ce instanceof ControllerClosedEvent) {
            setFailed();
        }


        if (ce instanceof ControllerEvent) {
            synchronized (getStateLock()) {
                getStateLock().notifyAll();
            }
        }
    }
}

public static void main(String args[]) {

}
```

}
<**Τέλος κώδικα**>

Σε αυτήν την κλάση έχουμε σχεδόν ότι έχουμε και στην προηγουμένη οπού κάνουμε transmit το video αυτή ξεκινάει όπως πριν από JmenuBar->Tools->Start Audio Stream.
Όπου ξεκινάει η ίδια ακριβώς σειρά γεγονότων εκτός του ορίσματος που πλέον είναι "dsound://" και ότι δεν ψάχνουμε track αλλά παίρνουμε το 1^{ο} διαθέσιμο.

Ανάλυση και Κώδικας Receiver Ήχου/Βίντεο

<**Αρχή κώδικα JChatVideoReceiver**>

```
package com.jchat.test.receiver;

import com.jchat.gui.PnlJChatStreamPlayer;
import java.io.*;
import java.awt.*;
import java.net.*;
import java.awt.event.*;
import java.util.*;

import javax.media.*;
import javax.media.rtp.*;
import javax.media.rtp.event.*;
import javax.media.rtp.rtcp.*;
import javax.media.protocol.*;
import javax.media.control.BufferControl;
import com.sun.media.rtp.RTPSessionMgr;
import javax.swing.JPanel;

public class JChatVideoReceiver implements ReceiveStreamListener,
SessionListener,
      ControllerListener, RemoteListener {

   Vector mgrs;
   private PlayerWindow playerWindow = null;
   private PnlJChatStreamPlayer streamPlayer = null;
   boolean dataReceived = false;
   Object dataSync = new Object();
   Vector targets;

   public JChatVideoReceiver(Vector targets) {
```

```
        this.targets = targets;
    }

    public void setStreamPlayer(PnlJChatStreamPlayer streamPlayer) {
        this.streamPlayer = streamPlayer;
    }

    public PlayerWindow getPlayerWindow() {
        return this.playerWindow;
    }

    public void initialize() throws Exception {
        System.out.println("Initialize...");
        mgrs = new Vector(targets.size());
        for (int i = 0; i < targets.size(); i++) {
            Target target = (Target) targets.elementAt(i);
            addTarget(target.ip, target.port, target.localPort);
        }
    }

    public long getBufferLength() {
        try {
            BufferControl control = (BufferControl) ((RTPManager)
mgrs.get(0)).getControl("javax.media.control.BufferControl");
            return control.getBufferLength();
        } catch (Exception ex) {
            return -1;
        }
    }

    public void setBufferLength(long length) {
        try {
            BufferControl control = (BufferControl) ((RTPManager)
mgrs.get(0)).getControl("javax.media.control.BufferControl");
            control.setBufferLength(length);
        } catch (Exception ex) {
        }
    }

    public void addTarget(String senderAddress,
            String senderPort,
            String localPort) {
        try {
```

```
        if (senderAddress.startsWith("/")) {
           senderAddress = senderAddress.substring(1);
        }
        InetAddress ipAddr;
        SessionAddress localAddr;
        SessionAddress destAddr;

        RTPManager mgr = RTPManager.newInstance();
        mgr.addSessionListener(this);
        mgr.addReceiveStreamListener(this);
        mgr.addRemoteListener(this);

        ipAddr = InetAddress.getByName(senderAddress);

        int local_port = new Integer(localPort).intValue();

        if (ipAddr.isMulticastAddress()) {
                localAddr = new SessionAddress(ipAddr,
                  local_port,
                  6);

           destAddr = new SessionAddress(ipAddr,
                  local_port,
                  6);
        } else {
           localAddr = new SessionAddress(InetAddress.getLocalHost(),
                  local_port);

           int remotePort = new Integer(senderPort).intValue();

           destAddr = new SessionAddress(ipAddr, remotePort);
        }

        mgr.initialize(localAddr);

        BufferControl bc = (BufferControl)
mgr.getControl("javax.media.control.BufferControl");

        if (bc != null) {
           bc.setBufferLength(350);
        }

        mgr.addTarget(destAddr);
```

```
            mgrs.addElement(mgr);
        } catch (Exception e) {
            e.printStackTrace();
            System.err.println("Cannot create the RTP Session: " + e.getMessage());
            return;
        }
    }

    public void close() {

        try {
            if (playerWindow != null) {
                playerWindow.close();
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        for (int i = 0; i < mgrs.size(); i++) {
            RTPManager mgr = (RTPManager) mgrs.elementAt(i);

            mgr.removeTargets("Closing session from AVReceiver");
            mgr.dispose();
            mgr = null;
        }
    }

    public synchronized void update(SessionEvent evt) {
        if (evt instanceof NewParticipantEvent) {
            Participant p = ((NewParticipantEvent) evt).getParticipant();
            System.err.println("  - A new participant had just joined: " +
p.getCNAME());
        }
    }

    public synchronized void update(ReceiveStreamEvent evt) {
        RTPManager mgr = (RTPManager) evt.getSource();

        Participant participant = evt.getParticipant();
        ReceiveStream stream = evt.getReceiveStream();

        String timestamp = getTimestamp();
```

```
        if (evt instanceof RemotePayloadChangeEvent) {

            System.err.println("  - Received an RTP PayloadChangeEvent.");
            System.err.println("Sorry, cannot handle payload change.");
            System.exit(0);

        } else if (evt instanceof NewReceiveStreamEvent) {
            try {
                stream = ((NewReceiveStreamEvent) evt).getReceiveStream();
                DataSource ds = stream.getDataSource();

                // Find out the formats.
                RTPControl ctl = (RTPControl)
ds.getControl("javax.media.rtp.RTPControl");
                if (ctl != null) {
                    System.err.println("  - Recevied new RTP stream: " +
ctl.getFormat());
                } else {
                    System.err.println("  - Recevied new RTP stream");
                }

                if (participant == null) {
                    System.err.println("      The sender of this stream had yet to be
identified.");
                } else {
                    System.err.println("      The stream comes from: " +
participant.getCNAME());
                }

                Player p = javax.media.Manager.createPlayer(ds);

                if (p == null) {
                    return;
                }

                p.addControllerListener(this);
                p.realize();
                playerWindow = new PlayerWindow(p, stream);
                synchronized (dataSync) {
                    dataReceived = true;
                    dataSync.notifyAll();
                }
```

```
            } catch (Exception e) {
                System.err.println("NewReceiveStreamEvent exception " +
e.getMessage());
                return;
            }
        } else if (evt instanceof StreamMappedEvent) {
            if (stream != null && stream.getDataSource() != null) {
                DataSource ds = stream.getDataSource();
                .
                RTPControl ctl = (RTPControl)
ds.getControl("javax.media.rtp.RTPControl");
                System.err.println("  - The previously unidentified stream ");


                if (ctl != null) {
                    System.err.println("      " + ctl.getFormat());
                }

                System.err.println("      had now been identified as sent by: " +
participant.getCNAME());

                RTPSessionMgr rtpManager = (RTPSessionMgr)
evt.getSessionManager();

                SessionAddress addr = rtpManager.getRemoteSessionAddress();

                if (addr != null) {
                    playerWindow.setTitle(addr.getDataHostAddress(),
                            addr.getDataPort());
                }
            }
        } else if (evt instanceof ByeEvent) {
            StringBuffer sb = new StringBuffer();

            sb.append(timestamp + " BYE");

            String reason = ((ByeEvent) evt).getReason();

            sb.append(" from " + participant.getCNAME());
            sb.append(" ssrc=" + stream.getSSRC());
            sb.append(" reason='" + reason + "'");

            if (playerWindow != null) {
```

```
            playerWindow.close();
        }
    }
}

public void update(RemoteEvent event) {
    String timestamp = getTimestamp();

    if (event instanceof ReceiverReportEvent) {
        ReceiverReport rr = ((ReceiverReportEvent) event).getReport();

        StringBuffer sb = new StringBuffer();

        sb.append(timestamp + " RR");

        if (rr != null) {
            Participant participant = rr.getParticipant();

            if (participant != null) {
                sb.append(" from " + participant.getCNAME());
                sb.append(" ssrc=" + rr.getSSRC());
            } else {
                sb.append(" ssrc=" + rr.getSSRC());
            }
        }
    } else if (event instanceof SenderReportEvent) {
        SenderReport sr = ((SenderReportEvent) event).getReport();

        StringBuffer sb = new StringBuffer();

        sb.append(timestamp + " SR");

        if (sr != null) {
            Participant participant = sr.getParticipant();

            if (participant != null) {
                sb.append(" from " + participant.getCNAME());
                sb.append(" ssrc=" + sr.getSSRC());
            } else {
                sb.append(" ssrc=" + sr.getSSRC());
            }
        }
    } else {
```

```
            System.out.println("RemoteEvent: " + event);
        }
    }

    private String getTimestamp() {
        String timestamp;

        Calendar calendar = Calendar.getInstance();

        int hour = calendar.get(Calendar.HOUR_OF_DAY);

        String hourStr = formatTime(hour);

        int minute = calendar.get(Calendar.MINUTE);

        String minuteStr = formatTime(minute);

        int second = calendar.get(Calendar.SECOND);

        String secondStr = formatTime(second);

        timestamp = hourStr + ":" + minuteStr + ":" + secondStr;

        return timestamp;
    }

    private String formatTime(int time) {
        String timeStr;

        if (time < 10) {
            timeStr = "0" + time;
        } else {
            timeStr = "" + time;
        }

        return timeStr;
    }

    public synchronized void controllerUpdate(ControllerEvent ce) {

        Player p = (Player) ce.getSourceController();

        if (p == null) {
```

```
            return;
        }
.
        if (ce instanceof RealizeCompleteEvent) {

            if (playerWindow == null) {
                System.err.println("Internal error!");
                System.exit(-1);
            }
            playerWindow.initialize();
            playerWindow.setVisible(true);
            p.start();
            streamPlayer.playStream(this);
        }

        if (ce instanceof ControllerErrorEvent) {
            p.removeControllerListener(this);
            if (playerWindow != null) {
                playerWindow.close();
            }

            System.err.println("AVReceiver internal error: " + ce);
        }
    }

    public class PlayerWindow extends JPanel {

        Player player;
        ReceiveStream stream;
        String senderPort;
        String senderAddress;

        PlayerWindow(Player p, ReceiveStream strm) {
            player = p;
            stream = strm;
        }

        public void setTitle(String address,
                int port) {
            senderAddress = address;
            senderPort = port + "";
        }
```

```
        public void initialize() {
            add(new PlayerPanel(player));
        }

        public void close() {
            player.close();
            setVisible(false);
        }

        public void addNotify() {
            super.addNotify();
        }
    }

    class PlayerPanel extends Panel {

        Component vc, cc;

        PlayerPanel(Player p) {
            setLayout(new BorderLayout());
            if ((vc = p.getVisualComponent()) != null) {
                add("Center", vc);
            }
        }

        public Dimension getPreferredSize() {
            return new Dimension(150, 150);
        }
    }

   public static void main(String args[]) {

    }
}
```

<**Τέλος κώδικα**>

Ο Receiver ξεκινάει με το initialize που παίρνει από το PnlJChatStreaming έτσι εκεί μέσω της addTarget προσθέτει τα στοιχειά του sender τα οποία είναι διεύθυνση και πόρτα και την δική μας πόρτα μέσω αυτόν δημιουργεί ένα νέο instance του RTP session manager όπως και στους transmitter ,διαβάζει την τοπική αλλά και απομακρυσμένη διεύθυνση (άλλου χρήστη) και την πόρτα (ελέγχει την παρουσία multicast address), έτσι κάνει initialize τον RTP session

manager , διαβάζει και τοποθετεί το μέγεθος του buffer ίσο με 350 και προσθέτουμε τον άλλο χρήστη στον RTP session manager.
Τώρα αναλόγως με το Event χρησιμοποιεί και την κατάλληλη update όπως προσθήκη νέου χρηστή, αλλαγές στο Payload το οποίο δεν έχουμε τοποθετήσει άρα θα πετάξει κάποιο error, ένα νέο stream όπου παίρνει το DataSource από αυτό το stream, βρίσκει το Format με τον RTPControl δημιουργεί τον Player τον φέρνει σε realized κατάσταση (όπου αυτόματα καλεί την controllerUpdate) και με τη χρήση της PlayerWindow (JPanel) τοποθετεί το Received stream στο πάνελ .
Υπάρχουν και άλλες update όπως η StreamMappedEvent που χρησιμοποιούμε για να αναγνωρίσουμε τους συμμετέχοντες ή η Receiver και SenderReportEvent

<**Αρχή κώδικα JChatAudioReceiver**>

```
package com.jchat.test.receiver;

import com.jchat.gui.PnlJChatStreamPlayer;
import java.io.*;
import java.awt.*;
import java.net.*;
import java.awt.event.*;
import java.util.*;

import javax.media.*;
import javax.media.rtp.*;
import javax.media.rtp.event.*;
import javax.media.rtp.rtcp.*;
import javax.media.protocol.*;
import javax.media.control.BufferControl;
import com.sun.media.rtp.RTPSessionMgr;

public class JChatAudioReceiver implements ReceiveStreamListener,
SessionListener,
      ControllerListener, RemoteListener {

   Vector mgrs;
   private PnlJChatStreamPlayer streamPlayer = null;
   boolean dataReceived = false;
   Object dataSync = new Object();
   Vector targets;

   public JChatAudioReceiver(Vector targets) {
      this.targets = targets;
   }
```

```
   public void setStreamPlayer(PnlJChatStreamPlayer streamPlayer) {
      this.streamPlayer = streamPlayer;
   }

   public void initialize() throws Exception {
      mgrs = new Vector(targets.size());
      for (int i = 0; i < targets.size(); i++) {
         Target target = (Target) targets.elementAt(i);
         addTarget(target.ip, target.port, target.localPort);
      }
   }

   public long getBufferLength() {
      try {
         BufferControl control = (BufferControl) ((RTPManager)
mgrs.get(0)).getControl("javax.media.control.BufferControl");
         return control.getBufferLength();
      } catch (Exception ex) {
         return -1;
      }
   }

   public void setBufferLength(long length) {
      try {
         BufferControl control = (BufferControl) ((RTPManager)
mgrs.get(0)).getControl("javax.media.control.BufferControl");
         control.setBufferLength(length);
      } catch (Exception ex) {
      }
   }

   public void addTarget(String senderAddress,
         String senderPort,
         String localPort) {
      try {
         if (senderAddress.startsWith("/")) {
            senderAddress = senderAddress.substring(1);
         }
         InetAddress ipAddr;
         SessionAddress localAddr;
         SessionAddress destAddr;
```

```
            RTPManager mgr = RTPManager.newInstance();
            mgr.addSessionListener(this);
            mgr.addReceiveStreamListener(this);
            mgr.addRemoteListener(this);

            ipAddr = InetAddress.getByName(senderAddress);

            int local_port = new Integer(localPort).intValue();

            if (ipAddr.isMulticastAddress()) {
               localAddr = new SessionAddress(ipAddr,
                     local_port,
                     6);

               destAddr = new SessionAddress(ipAddr,
                     local_port,
                     6);
            } else {
               localAddr = new SessionAddress(InetAddress.getLocalHost(),
                     local_port);

               int remotePort = new Integer(senderPort).intValue();

               destAddr = new SessionAddress(ipAddr, remotePort);
            }

            mgr.initialize(localAddr);

            BufferControl bc = (BufferControl)
mgr.getControl("javax.media.control.BufferControl");

            if (bc != null) {
               bc.setBufferLength(350);
            }

            mgr.addTarget(destAddr);

            mgrs.addElement(mgr);
         } catch (Exception e) {
            e.printStackTrace();
            System.err.println("Cannot create the RTP Session: " + e.getMessage());
            return;
         }
```

```
    }

    public void close() {
        for (int i = 0; i < mgrs.size(); i++) {
            RTPManager mgr = (RTPManager) mgrs.elementAt(i);

            mgr.removeTargets("Closing session from AVReceiver");
            mgr.dispose();
            mgr = null;
        }
    }

    public synchronized void update(SessionEvent evt) {
        if (evt instanceof NewParticipantEvent) {
            Participant p = ((NewParticipantEvent) evt).getParticipant();
            System.err.println("  - A new participant had just joined: " +
p.getCNAME());
        }
    }

        public synchronized void update(ReceiveStreamEvent evt) {
        RTPManager mgr = (RTPManager) evt.getSource();

        Participant participant = evt.getParticipant();         // could be null.
        ReceiveStream stream = evt.getReceiveStream();  // could be null.

        String timestamp = getTimestamp();

        if (evt instanceof RemotePayloadChangeEvent) {

            System.err.println("  - Received an RTP PayloadChangeEvent.");
            System.err.println("Sorry, cannot handle payload change.");
            System.exit(0);

        } else if (evt instanceof NewReceiveStreamEvent) {
            try {
                stream = ((NewReceiveStreamEvent) evt).getReceiveStream();
                DataSource ds = stream.getDataSource();

                RTPControl ctl = (RTPControl)
ds.getControl("javax.media.rtp.RTPControl");
                if (ctl != null) {
```

```
                System.err.println("  - Recevied new RTP stream: " +
ctl.getFormat());
            } else {
                System.err.println("  - Recevied new RTP stream");
            }

            if (participant == null) {
                System.err.println("      The sender of this stream had yet to be
identified.");
            } else {
                System.err.println("      The stream comes from: " +
participant.getCNAME());
            }

            Player p = javax.media.Manager.createPlayer(ds);

            if (p == null) {
                return;
            }

            p.addControllerListener(this);
            p.realize();
            synchronized (dataSync) {
                dataReceived = true;
                dataSync.notifyAll();
            }

        } catch (Exception e) {
            System.err.println("NewReceiveStreamEvent exception " +
e.getMessage());
            return;
        }
    } else if (evt instanceof StreamMappedEvent) {
        if (stream != null && stream.getDataSource() != null) {
            DataSource ds = stream.getDataSource();
            RTPControl ctl = (RTPControl)
ds.getControl("javax.media.rtp.RTPControl");
            System.err.println("  - The previously unidentified stream ");


            if (ctl != null) {
                System.err.println("      " + ctl.getFormat());
            }
```

```
            System.err.println("      had now been identified as sent by: " +
participant.getCNAME());

            RTPSessionMgr rtpManager = (RTPSessionMgr)
evt.getSessionManager();

            SessionAddress addr = rtpManager.getRemoteSessionAddress();
         }
      } else if (evt instanceof ByeEvent) {
      }
   }

   public void update(RemoteEvent event) {
      String timestamp = getTimestamp();

      if (event instanceof ReceiverReportEvent) {
         ReceiverReport rr = ((ReceiverReportEvent) event).getReport();

         StringBuffer sb = new StringBuffer();

         sb.append(timestamp + " RR");

         if (rr != null) {
            Participant participant = rr.getParticipant();

            if (participant != null) {
               sb.append(" from " + participant.getCNAME());
               sb.append(" ssrc=" + rr.getSSRC());
            } else {
               sb.append(" ssrc=" + rr.getSSRC());
            }
         }
      } else if (event instanceof SenderReportEvent) {
         SenderReport sr = ((SenderReportEvent) event).getReport();

         StringBuffer sb = new StringBuffer();

         sb.append(timestamp + " SR");

         if (sr != null) {
            Participant participant = sr.getParticipant();
```

```
            if (participant != null) {
                sb.append(" from " + participant.getCNAME());
                sb.append(" ssrc=" + sr.getSSRC());
            } else {
                sb.append(" ssrc=" + sr.getSSRC());
            }
        }
    } else {
        System.out.println("RemoteEvent: " + event);
    }
}

private String getTimestamp() {
    String timestamp;

    Calendar calendar = Calendar.getInstance();

    int hour = calendar.get(Calendar.HOUR_OF_DAY);

    String hourStr = formatTime(hour);

    int minute = calendar.get(Calendar.MINUTE);

    String minuteStr = formatTime(minute);

    int second = calendar.get(Calendar.SECOND);

    String secondStr = formatTime(second);

    timestamp = hourStr + ":" + minuteStr + ":" + secondStr;

    return timestamp;
}

private String formatTime(int time) {
    String timeStr;

    if (time < 10) {
        timeStr = "0" + time;
    } else {
        timeStr = "" + time;
    }
```

```
        return timeStr;
    }

    public synchronized void controllerUpdate(ControllerEvent ce) {

        Player p = (Player) ce.getSourceController();

        if (p == null) {
            return;
        }
        if (ce instanceof RealizeCompleteEvent) {
            p.start();

        }

        if (ce instanceof ControllerErrorEvent) {
            p.removeControllerListener(this);
        }
    }

}
```

<**Τέλος κώδικα**>

Εδώ έχουμε την ιδία κλάση όπως την JChatVideoReceiver όπου καλείται και λειτουργεί με τον ίδιο τρόπο χωρίς όμως να υπάρχει ανάγκη για δημιουργία κάποιου πάνελ για την εμφάνιση εφόσον το χρησιμοποιούμε για ήχο.

<**Αρχη κωδικα Target**>

```
package com.jchat.test.receiver;

public class Target {
    String localPort;
    String ip;
    String port;

    public Target( String localPort,
                   String ip,
                   String port) {
        this.localPort= localPort;
        this.ip= ip;
        this.port= port;
```

```
    }
}
```

<**Τέλος κώδικα**>

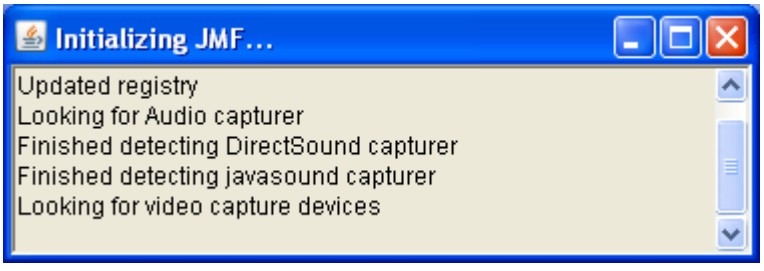
Αυτή είναι η κλάση για την αποθήκευση στοιχειών χρηστών για τη χρήση τους με την μετάδοση και λήψη ήχου/βίντεο όπου αποθηκεύονται η τοπική μας πόρτα , η IP και η πόρτα του αλλού συμμετέχων.

6. Βιβλίο Οδηγιών Χρήστη

Για να μπορείτε να χρησιμοποιήσετε το Πρόγραμμα αλλά και να μπορείτε να έχετε πρόσβαση σε οτιδήποτε αφορά την JMF πρέπει να έχετε κάνει κάποια βήματα που αφορούν την εγκατάσταση και παραμετροποίηση όσο αφορά τα PATH τα οποία πρέπει να υπάρχουν, εδώ θα βρείτε οδηγίες για όλα αυτά συν τον τρόπο χρήσης του προγράμματος.

Εγκατάσταση JMF

Υποθέτοντας ότι έχετε εγκατεστημένο το πρόσφατο JRE (Java Runtime Environment)
(αν όχι θα το βρείτε http://java.sun.com/javase/downloads/index.jsp).
Κατεβάστε το Windows Performance pack JMF 2.1.1e (ή πιο πρόσφατο)
Από την σελίδα http://java.sun.com/products/java-media/jmf/2.1.1/download.html θα βρείτε το JMF 2.1.1e.
Ξεκινήστε την εγκατάσταση και ακολουθήστε τις οδηγίες για την εγκατάσταση του αφού πολύ γρήγορα τελειώσει η εγκατάσταση θα εμφανίσει ένα παράθυρο όπως το παρακάτω με το οποίο όπως βλέπουμε διαβάζει συσκευές ήχου και βίντεο, μόλις τελειώσει θα σας ζητήσει να επανεκκινήσετε τον υπολογιστή σας. Παρακαλώ κάντε το.



Εικόνα 3

Μόλις ξεκινήσει ο υπολογιστής ανοιχτέ ένα command prompt (Έναρξη->Εκτέλεση, και μέσα στην περιοχή γράφουμε cmd και πατάμε Enter).

Στην εγκατάσταση αυτόματα το Performance pack θα μας έχει εγκαταστήσει τα σωστά PATH και CLASSPATH αλλά για να είμαστε σίγουρη μπορούμε να πάμε στην παρακάτω ιστοσελίδα
http://java.sun.com/products/java-media/jmf/2.1.1/jmfdiagnostics.html

Αν μας εμφανίσει ότι οι βιβλιοθήκες δεν βρεθήκαν (Native Libraries Not Found) απλά κάντε επικόληση τις παρακάτω γραμμες μέσα στο command prompt (με την προϋπόθεση ότι έχουμε το jmf 2.1.1e).

Set JMFDIR=c:\Program Files\JMF2.1.1e
set CLASSPATH=%JMFDIR%\lib\jmf.jar;%JMFDIR%\lib\sound.jar;.;%CLASSPATH%
set PATH=%JMFDIR%\lib;%PATH%
 και για να είστε σίγουροι ότι όλα πήγαν καλά μπορείτε να χρησιμοποιήσετε την ιστοσελίδα ξανά.
http://java.sun.com/products/java-media/jmf/2.1.1/jmfdiagnostics.html

Όπου θα εμφανίσει Αυτό

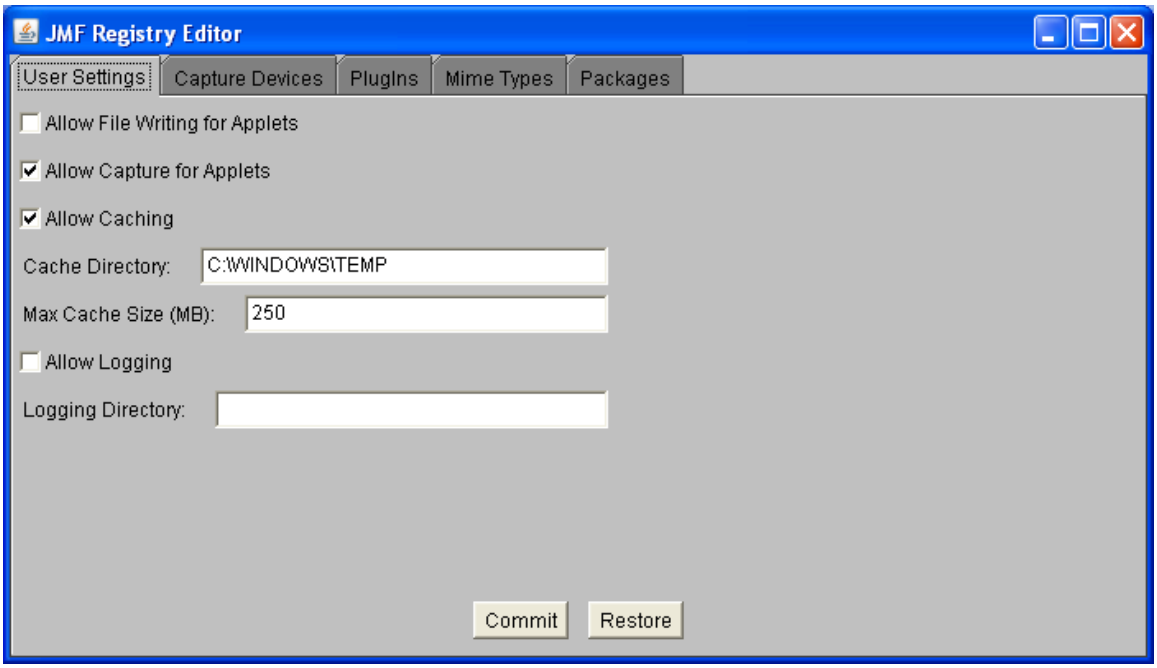
Εικόνα 4

Αυτό σημαίνει ότι όλα πήγαν καλά , αν όχι επαναλάβετε τις παραπάνω διαδικασίες διότι κάτι δεν πήγε καλά.

Εγκατάσταση νέου υλικού

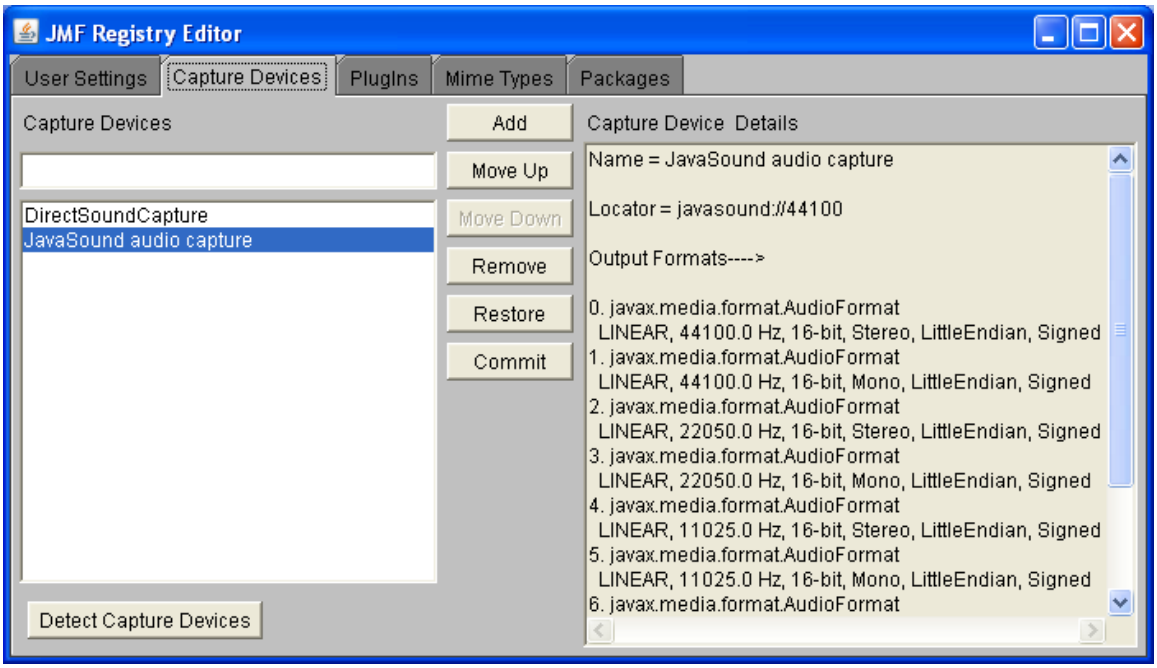
Αν όταν κάνατε την εγκατάσταση δεν είχατε συνδεδεμένο το υλικό για να το εγκαταστήσει αυτόματα ή αν απλά έχετε κάποιο άλλο νέο ή παλιό υλικό και θέλετε να χρησιμοποιήσετε αυτό πρέπει να κάνετε κάποιες ρυθμίσεις για να το χρησιμοποιήσετε με την JMF.

Πηγαίνετε στο Έναρξη->Προγράμματα->Java Media Framework 2.1.1e->JMF Registry και θα εμφανίσει αυτό το παράθυρο.



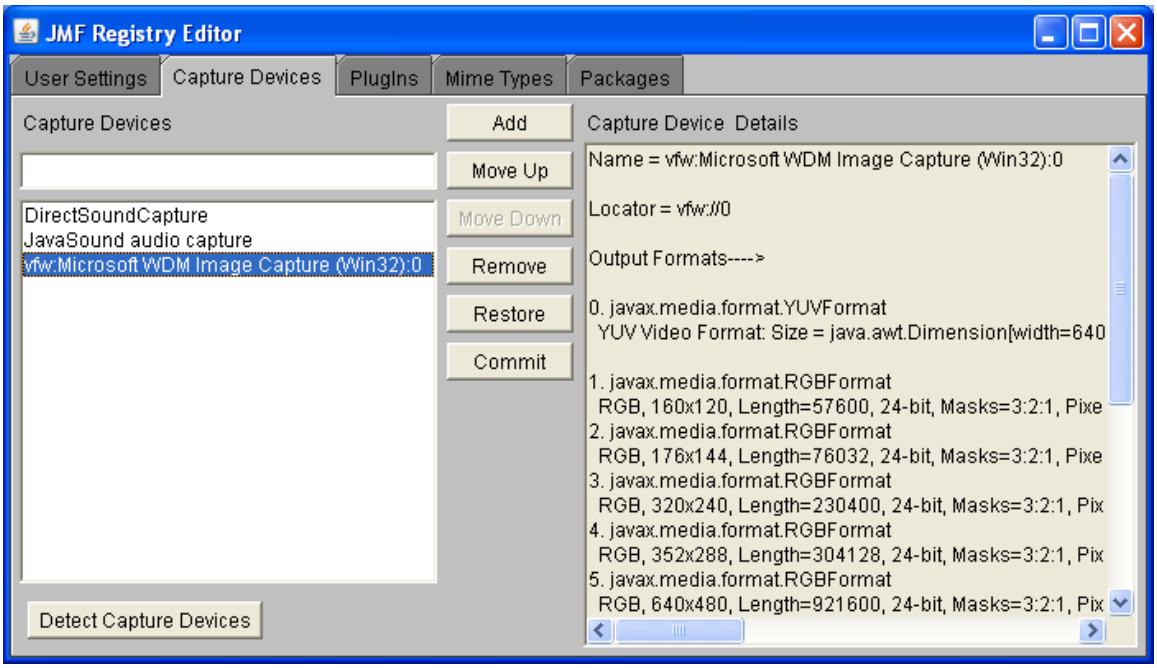
Εικόνα 5

Εμείς πρέπει να πάμε στο Capture Devices ώστε να αναγνωρίσει την συσκευή μας. Θα δούμε κάτι σαν αυτό.



Εικόνα 6

Αφού όμως πατήσουμε το Detect Capture Devices και περιμένουμε μερικά λεπτά θα δούμε αυτό.



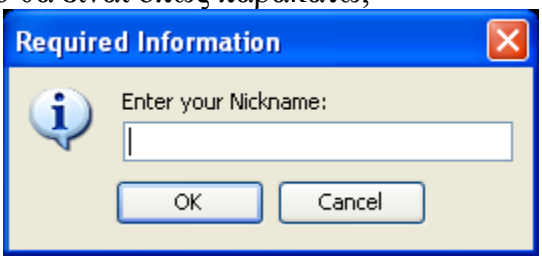
Εικόνα 7

Προσέξτε το Επιλεγμένο αντικείμενο είναι η νέα κάμερα που αναγνώρισε, αφού την αναγνώρισε πρέπει να πατήσουμε στο Commit ώστε να αποθήκευση τις αλλαγές να το κλείσουμε και να κάνουμε μια επαννεκίνηση, τώρα όλα είναι έτοιμα.

Χρήση του Προγράμματος

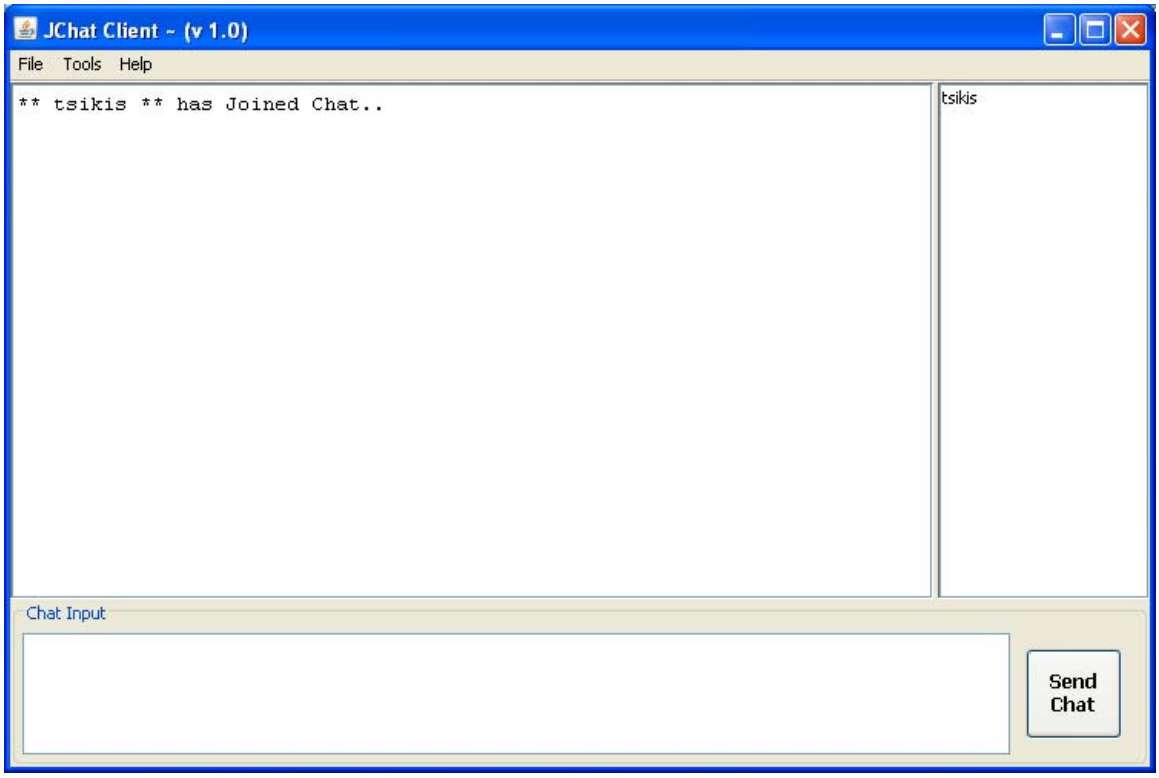
Αφού έχουμε πλέον όλα έτοιμα μπορούμε να ασχοληθούμε με το κάθε αυτό πρόγραμμα, Έτσι υπάρχουν 2 αρχεία bat το ένα για το server και το άλλο για τους χρηστές με ονόματα server.bat και client.bat αντίστοιχα το server.bat το τρέχει μόνο αυτός ο οποίος θα παρέχει το server για την επικοινωνία αντίθετα το άλλο αρχείο το τρέχουν όλοι όσοι θέλουν να συνδεθούν.
Για τον server είναι απλά τα πράγματα απλά το τρέχουμε και έχουμε τον server μας.
Όσο αφορά τους χρήστες υπάρχουν 2 τρόποι είτε από command prompt πηγαίνουμε στον φάκελο που έχουμε τα εκτελέσιμα και τρέχουμε client.bat <IP του server> (χωρίς τα εισαγωγικά) είτε απλά ανοίγουμε το αρχείο client.bat και αλλάζουμε το %1 με την IP του server (το οποίο δεν συνιστάται γιατί ο server μπορεί να αλλάζει IP).

Αφού γίνει αυτό θα μας εμφανίστε ένα παράθυρο που θα ζητάει το nickname μας ένα όνομα χρήστη που θα είναι όπως παρακάτω,



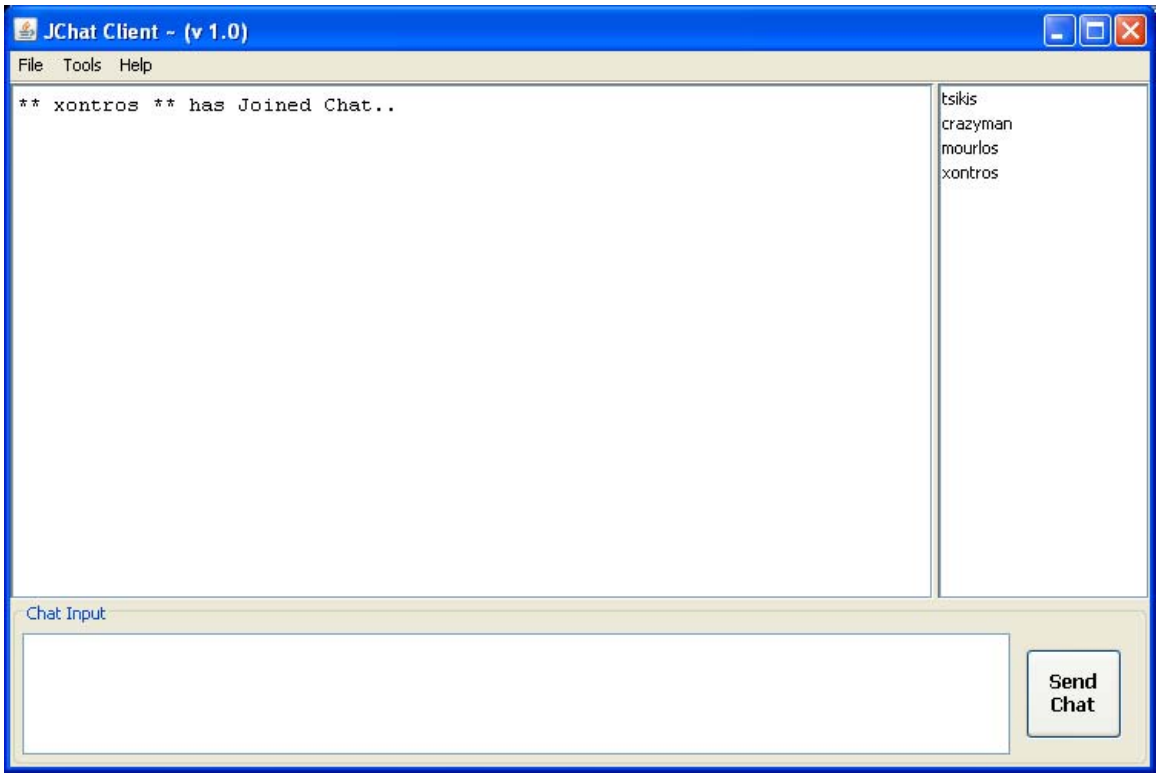
Εικόνα 8

Αφού βάλουμε το όνομα χρήστη που θέλουμε θα μας εμφανίσει αυτό,



Εικόνα 9

Αφού μπουν και άλλοι χρήστες η λίστα θα ανανεώνεται και θα προστίθενται και άλλοι χρήστες.



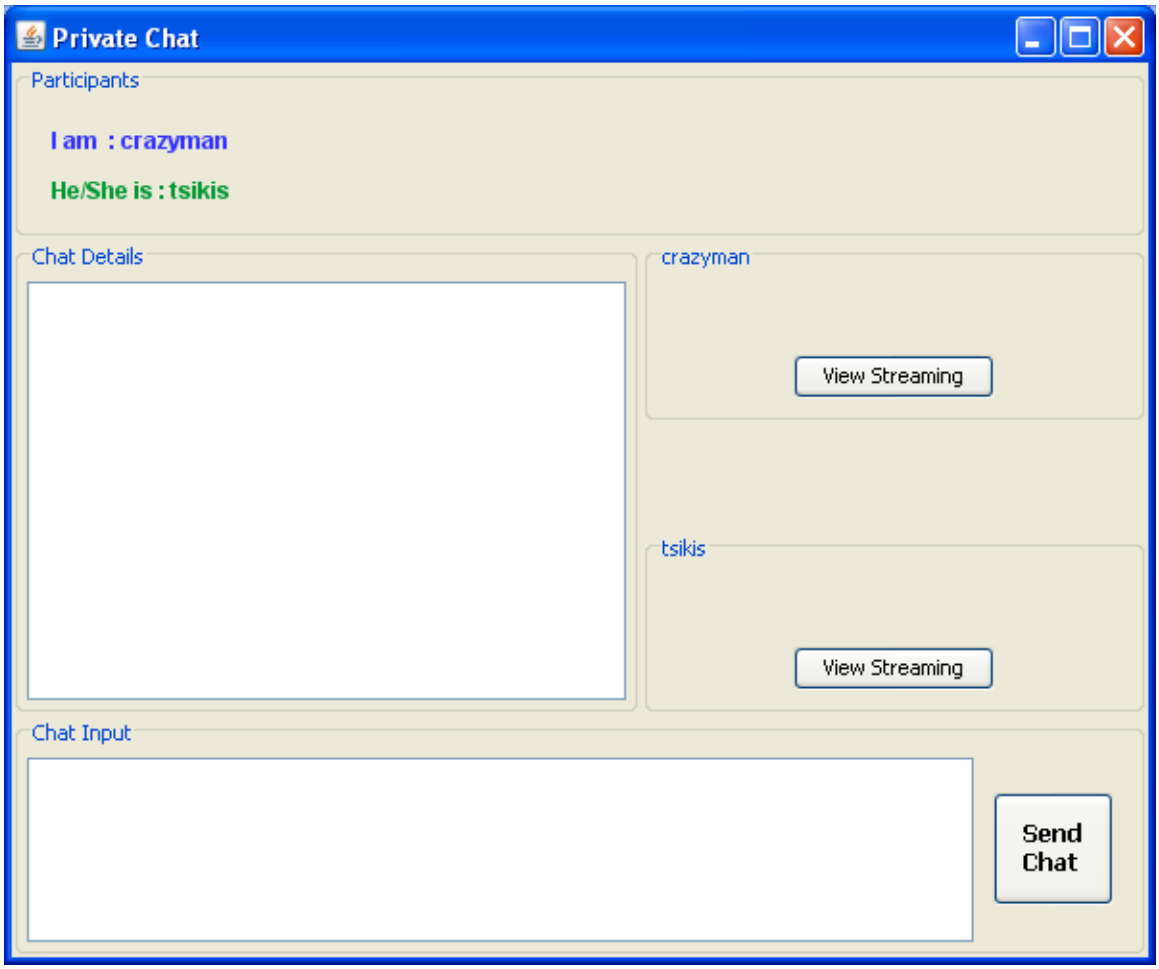
Εικόνα 10

Σε αυτήν την κατάσταση μπορούμε να γράψουμε και να μας δουν όλοι οι χρήστες την συγκεκριμένη στιγμή.
Για την επικοινωνία μέσω βίντεο και/ή ήχου πηγαίνουμε στην μπάρα του μενού και επιλέγουμε το Tools όπου θα εμφανιστούν 2 επιλογές οι οποίες είναι:

- Start My Webcam
- Start Audio Stream

Η πρώτη επιλογή ξεκινάει την κάμερα μας ενώ η δεύτερη τον ήχο αν υπάρξουν προβλήματα εδώ σίγουρα δεν έχετε κάνει σωστά τα βήματα 6.1 ή/και 6.2 παρακαλώ κοιτάξτε τα ξανά.

Αφού ξεκινήσετε ότι χρειάζεστε για την επικοινωνία σας μπορείτε να ανοίξετε ένα άλλο παράθυρο για προσωπική επικοινωνία με ένα άλλο χρήστη με ένα απλό κλικ στο όνομα του. Έτσι θα ανοίξει αυτό το παράθυρο



Εικόνα 11

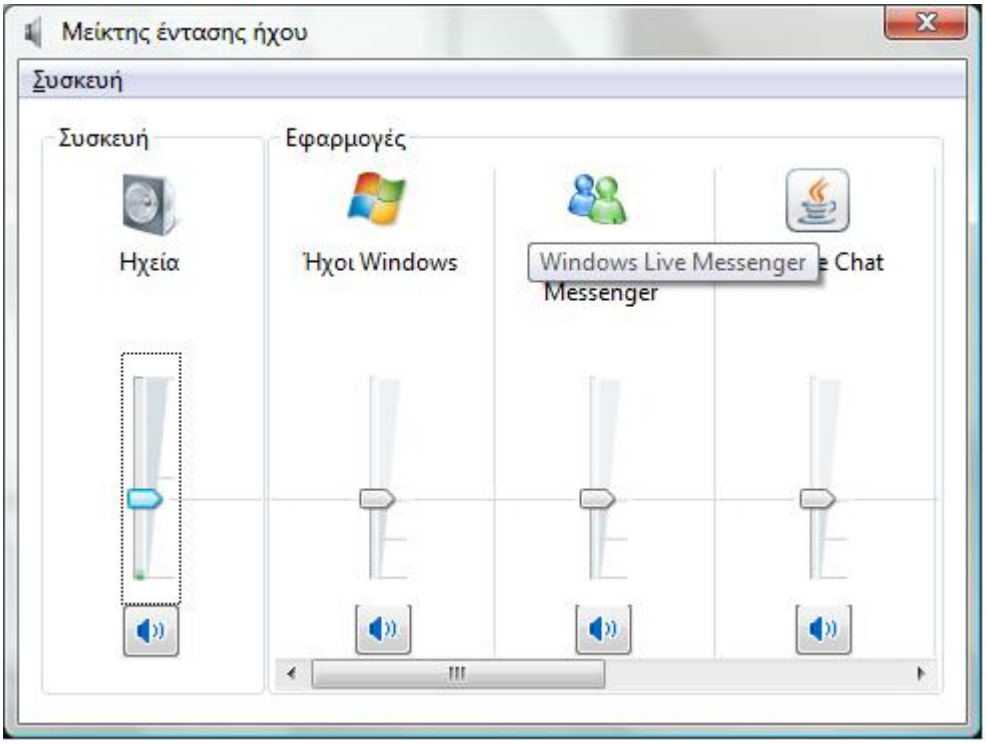
Όπου αρχικά θα το βλέπει μόνο αυτός ο οποίος άνοιξε την προσωπική συζήτηση μόλις ο χρήστης στείλει ένα μήνυμα και με την προϋπόθεση ότι ο άλλος χρήστης έχει ενεργοποιήσει την κάμερα του ή/και τον Ήχο θα δείτε αυτό και από τις δυο πλευρές αφού έχετε πατήσει το View Streaming Έτσι ώστε να βλέπετε ο Ένας τον άλλον.

Εικόνα 12

Τώρα σε αυτό το σημείο με την προϋπόθεση ότι έχετε ενεργοποιήσει και οι δυο ότι χρειάζεστε βλέπετε και ακούτε ο Ένας τον άλλον, ενώ παράλληλα μπορείτε να

γράφετε ο Ένας στον άλλον για πιο προσωπικά δεδομένα η για οπότε το θεωρείτε απαραίτητο.
Πολλές φορές λόγω εύρους ζώνης, κακών γραμμών, κακών καιρικών συνθηκών, η και πολλών άλλων λόγων και στοιχείων μπορεί να υπάρχουν καθυστερήσεις είτε στο Ήχο είτε στο βίντεο, Έτσι υπάρχει όπως βλέπουμε ένας έλεγχος μέσω των + και – κουμπιών που βλέπουμε και αφού επιλέξουμε τι θέλουμε να αλλάξουμε το Audio η το Video Buffer, όσο το μεγαλώνουμε τόσο μεγαλώνει το buffer όσο το μικραίνουμε τόσο μικραίνει, Έτσι αναλόγως με την περίπτωση μας το αλλάζουμε.

Ακόμη λόγω της χρήσης JMF και JAVA υπάρχουν και περεταίρω δυνατότητες, για παράδειγμα σε λειτουργικό σύστημα όπως τα Vista λόγω της JAVA μπορούμε να χειριστούμε ξεχωριστά την ένταση του ήχου όπως βλέπουμε παρακάτω.



Εικόνα 13

Αφού έχουμε τελειώσει την επικοινωνία απλά κλείνουμε το παράθυρο , σταματάμε αντίστοιχα ότι είχαμε ενεργοποιήσει προηγουμένως όπως την κάμερα η/και τον Ήχο και μπορούμε να κλείσουμε το πρόγραμμα.

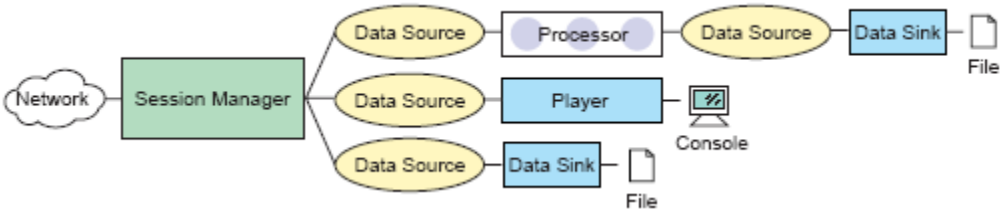
7. Βιβλίο Οδηγιών Προγραμματιστή

Η JMF και γενικά η JAVA περιλαμβάνει μια πληθώρα μεθόδων που μπορούν να χρησιμοποιηθούν για την περαιτέρω διευκόλυνση μας αλλά και την επίλυση κάποιων προβλημάτων που ίσως να υπάρξουν.

Εδώ θα ασχοληθουμε με την JMF και καποιες μεθοδους της τις οποιες μπορουμε να χρησιμοποιησουμε, να προσθεσουμε, να αλλαξουμε, να τροποποιησουμε.

Η JMF επιτρέπει την αναπαραγωγή και την εκπομπή RTP Stream και μπορεί να υποστηρίξει επιπλέον RTP Formats και δυναμικά payloads μέσω των διαφόρων plugin μηχανισμών της.

Μπορούμε να αναπαραγάγουμε RTP Streams τοπικά να τα σώσουμε σε ένα αρχείο ή και τα δυο.



Εικόνα 14

Στην JMF υπάρχει ο SessionManager ο οποίος χειρίζεται τις RTP συνεδρίες. Ακόμη θυμάται τους συμμετέχοντες και τα Streams τα οποία εκπεμφθήκαν.

Ακόμη ο SessionManager κρατάει στατιστικά από όλα τα RTP και RTCP πακέτα που σταλθήκαν και λήφθηκαν, τα οποία μπορούν να παρακολουθούνται σε ολόκληρη τη συνεδρία ή ανά Stream, τα στατιστικά τα οποία μπορούμε να χρησιμοποιήσουμε για δικούς μας λογούς είναι :

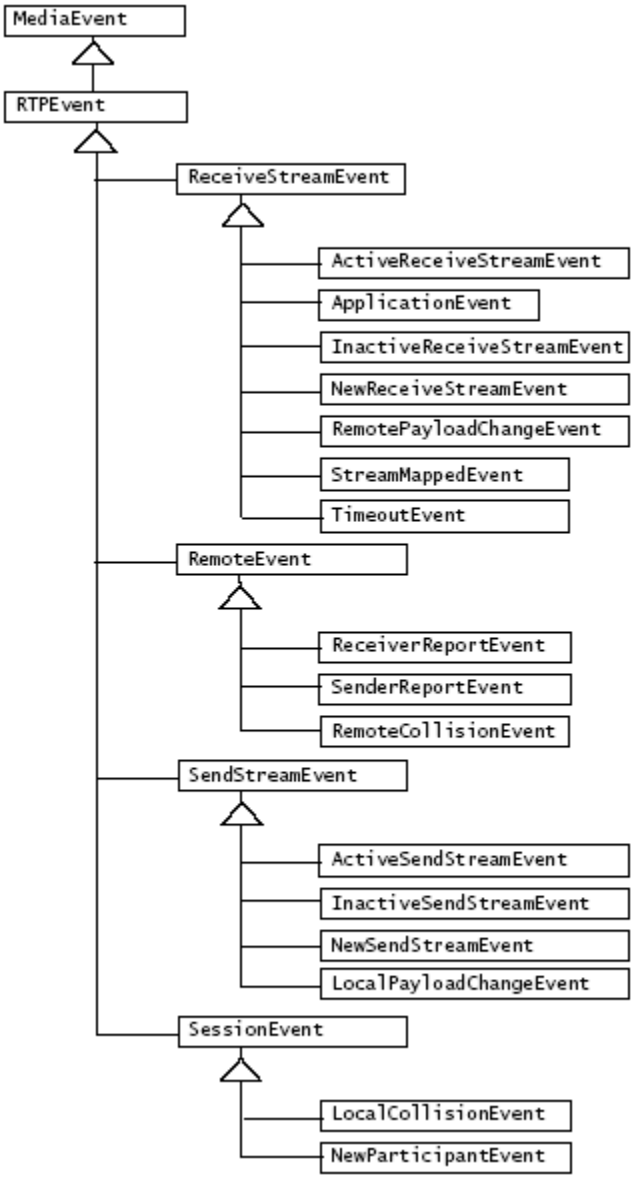
- GlobalReceptionSats: Κρατάει συνολικά ληφθέντα στατιστικά της συνεδρίας
- GlobalTransmissionStats: Κρατάει συνολικά στατιστικά από όλους του χρήστες που έκπεμψαν στη συνεδρία
- ReceptionStats: Κρατάει στατιστικά που ληφθήκαν ανά συμμετέχοντα στη συνεδρία
- TransmissionStats: Κρατάει στατιστικά που έκπεμψαν ξεχωριστά ο κάθε συμμετέχοντας

Ο SessionManager κρατάει ένα αντικείμενο RTPStream για κάθε Stream στην συνεδρία, υπάρχουν 2 είδη RTP Stream:

- ReceiveStream που σημαίνει το Stream που λαμβάνουμε από άλλο συμμετέχοντα.
- SendStream που σημαίνει ένα Stream δεδομένων που έρχεται από τον Processor ή DataSource και στέλνεται μέσω του Δικτύου.

RTP Events

Υπάρχουν κάποια RTP Events τα οποία χρησιμοποιούνται για την παρακολούθηση της κατάστασης μια συνεδρίας αλλά και Stream ξεχωριστά.



Εικόνα 15

Για την λήψη ειδοποιήσεων των RTP Events πρέπει να χρησιμοποιήσουμε τους κατάλληλους listener και να τους κάνουμε register στον SessionManager.

- SessionListener: Δέχεται ειδοποιήσεις για τις αλλαγές στην κατάσταση της συνεδρίας.
- SendStreamListener: Δέχεται ειδοποιήσεις για τις αλλαγές στην κατάσταση των Stream που εκπέμπονται.
- ReceiveStreamListener: Δέχεται ειδοποιήσεις για τις αλλαγές στην κατάσταση των Stream που λαμβάνουμε.
- RemoteListener: Δέχεται ειδοποιήσεις για Events ή RTP μηνύματα ελέγχου που λήφθηκαν από ένα απομακρυσμένο συμμετέχοντα.

Session Listener

Μπορούμε να χρησιμοποιήσουμε τον SessionListener να λαμβάνει ειδοποιήσεις για events που αφορούν την RTP συνεδρία όπως η προσθήκη νέων συμμετεχόντων.
Υπάρχουν 2 είδη τέτοιων event:

- NewParticipantEvent: Δείχνει ότι ένας νέος χρήστης μπήκε στην συνεδρία.
- LocalCollisionEvent: Δείχνει ότι η πηγή συγχρονισμού είναι ήδη σε χρήση.

Send Stream Listener

Μπορούμε να χρησιμοποιήσουμε τον SendStreamListener για να παίρνουμε ειδοποιήσεις οπουδήποτε

- Νέα Send Streams δημιουργούνται με κάθε νέο συμμετέχοντα
- Μπορούμε να μάθουμε αν η μεταφορά των δεδομένων από την πηγή που δημιούργησε το Stream ξεκίνησε ή σταμάτησε
- Τα Format του Stream.

Υπάρχουν 5 τύποι από γεγονότα που αφορούν ένα SendStream:

- NewSendStreamEvent: Δείχνει ότι ένα νέο Stream δημιουργήθηκε από τον συμμετέχοντα.
- ActiveSendStreamEvent:Δείχνει ότι η μεταφορά των δεδομένων από την πηγή έχει ξεκινήσει
- InactiveSendStreamEvent: Δείχνει ότι η μεταφορά των δεδομένων από την πηγή έχει σταματήσει.
- LocalPayloadChangeEvent: Δείχνει ότι το Format ή Payload έχει αλλάξει.
- StreamClosedEvent: Δείχνει ότι το Stream έχει κλείσει.

Receive Stream Listener

Μπορούμε να χρησιμοποιήσουμε τον ReceiveStreamListener για να λαμβάνουμε ειδοποιήσεις όταν:

- Νέα receive stream έχουν δημιουργηθεί
- Η μεταφορά δεδομένων έχει ξεκινήσει η έχει σταματήσει
- Η μεταφορά των δεδομένων έχει ένα Time-out
- Όταν ένα προηγουμένως «ορφανό» ReceiveStream έχει πλέον ονομαστεί από κάποιον Συμμετέχοντα.
- Έχουμε λάβει ένα RTCP APP πακέτο.
- Το Format ή το Payload έχει του receive stream έχει αλλάξει.

Μπορούμε ακόμη να χρησιμοποιήσομε το REceiveSTreamListener για να έχουμε πρόσβαση στην RTP πηγή δεδομένων ούτως ώστε να ξεκινήσουμε ένα MediaHandler.

Υπάρχουν 7 τύποι γεγονότων για ReceiveStream:

- NewReceiveStreamEvent: Δείχνει ότι ο Session Manager έχει δημιουργήσει ένα νέο receive stream για μια νέο-εντοπισμένη πηγή.
- ActiveReceiveStreamEvent: Δείχνει ότι η μεταφορά δεδομένων έχει ξεκινήσει.
- InactiveReceiveStreamEvent: Δείχνει ότι η μεταφορά δεδομένων έχει σταματήσει.
- TimeoutEvent: Δείχνει ότι η μεταφορά των δεδομένων έχει Timeout.
- RemotePayloadChangeEvent: Δείχνει ότι το Format ή το Payload της λαμβάνουσας stream έχει αλλάξει.
- StreamMappedEvent: Δείχνει ότι ένα προηγουμένως <ορφανό> λάμβαναν stream έχει πλέον αναγνωριστεί με ένα συμμετέχοντα.
- ApplicationEvent: Δείχνει ότι ένα RTCP APP πακέτο έχει ληφθεί.

Remote Listener

Μπορούμε να χρησιμοποιήσουμε τον RemoteListener για να λαμβάνουμε πληροφορίες γεγονότων ή RTP μηνυμάτων ελέγχου από ένα απομακρυσμένο συμμετέχοντα. Μπορεί να χρησιμοποιήσετε τον RemoteListener σε μια εφαρμογή για να ελέγχουμε την κατάσταση της συνεδρίας-μας επιτρέπει να λαμβάνουμε RTCP αναφορές και να ελέγχουμε την ποιότητα της συνεδρίας που δεχόμαστε χωρίς να χρειάζεται να λαμβάνουμε δεδομένα ή πληροφορίες για κάθε Stream.

Υπάρχουν 3 τύποι γεγονότων για ένα απομακρυσμένο συμμετέχοντα.

- ReceiveReportEvent: Δείχνει ότι ένα RTP μήνυμα παραλήπτη αναφοράς έχει ληφθεί.
- SenderReportEvent: Δείχνει ότι ένα RTP μήνυμα αποστολέα έχει ληφθεί.
- RemoteCollisionEvent: Δείχνει ότι 2 διαφορετικοί συμμετέχοντες χρησιμοποιούν την ιδία πηγή συγχρονισμού.

RTP Data

Τα Streams μέσα σε μια RTP συνεδρία παρουσιάζονται από RTPStream αντικείμενα.Υπάρχουν 2 τύποι RTPStreams:ReceiveStream και SendStream. Κάθε RTP Stream έχει ένα buffer για την πηγή δεδομένων για αυτόν.Για ReceiveStreams χρησιμοποιούμε πάντα τον PushBufferDataSource.

Ο Session manager αυτόματα κατασκευάζει ένα νέο receive stream όταν αντιλαμβάνεται ότι νέα streams έρχονται από απομακρυσμένους συμμετέχοντες. Μπορούμε να κατασκευάσουμε νέα Send streams καλώντας την createSendStream του session manager.

Data Handlers

Τα JMF RTP APIs είναι σχεδιασμένα ώστε να είναι ανεξάρτητα από το πρωτόκολλο μεταφοράς. Μπορούμε όμως να κατασκευάσουμε ένα RTP data handler ούτως ώστε να επιτρέψουμε στο JMF να λειτουργήσει πάνω από ένα συγκεκριμένο πρωτόκολλο μεταφοράς. Αυτός ο data handler είναι το DataSource και μπορεί να χρησιμοποιηθεί ως η media source για τον player.

Η αφηρημένη κλάση RTPPushDataSource είναι το βασικό στοιχείο για ένα RTP data handler. Ένας data handler έχει και stream λήψης δεδομένων (PushSourceStream) αλλά και stream αποστολής δεδομένων (outputDataStream). Ο data handler μπορεί να χρησιμοποιηθεί είτε από το κανάλι δεδομένων είτε από το κανάλι ελέγχου μια RTP συνεδρίας.

Ένα RTPSocket είναι ένα RTPPushDataSource και έχει και κανάλι δεδομένων και κανάλι ελέγχου δεδομένων. Κάθε κανάλι έχει ένα stream λήψης και αποστολής για να κάνει stream δεδομένα σε και από κάποιο δίκτυο. Ένα RTPSocket μπορεί να έχει RTPControls για να περνάει δυναμικό payload στον session manager.

RTP Data Formats

Όλα τα RTP δεδομένα χρησιμοποιούν RTP Format κωδικοποίηση AudioFormat και VideoFormat κλάσης. Για παράδειγμα ένα Jpeg κωδικοποιημένο video έχει την κωδικοποίηση δηλωμένη ως VideoFormat.JPEG_RTP.

Τα AudioFormat έχουν 4 κωδικοποιήσεις.

```
public static final String ULAW_RTP = "JAUDIO_G711_ULAW/rtp";
public static final String DVI_RTP = "dvi/rtp";
public static final String G723_RTP = "g723/rtp";
public static final String GSM_RTP = "gsm/rtp";
```

Ενώ το VideoFormat έχει 3 κωδικοποιήσεις.

```
public static final String JPEG_RTP = "jpeg/rtp";
public static final String H261_RTP = "h261/rtp";
public static final String H263_RTP = "h263/rtp";
```

RTP Controls

ΤΟ RTP API χρησιμοποιεί ένα RTP έλεγχο RTPControl.RTPControl. Μας δίνει μηχανισμούς για να προσθέσουμε μια ένωση μεταξύ του δυναμικού Payload και του Format.RTPControl ακόμη παρέχει μεθόδους για να χρησιμοποιήσουμε στατιστικά της συνεδρίας και να δούμε το παρόν Payload και Format.

Ο SessionManager ακόμη παρέχει Controls που επιτρέπει σε μια ένα session manager να χρησιμοποιήσει ακόμα περισσοτέρους ελέγχους μέσω των getControl και getControls μεθόδων. Για παράδειγμα ο session manager μας επιτρέπει να έχουμε ένα BufferControl και να ελέγχουμε το μέγεθος του buffer.

Reception

Η παρουσία ενός εισερχομένου RTP Stream χειρίζεται από ένα Player. Για να πάρουμε και να παρουσιάσουμε ένα stream από μια RTP συνεδρία μπορούμε να χρησιμοποιήσουμε ένα MediaLocator ο οποίος θα κατασκευάσει τον Player. Ένας media locator για μια RTP συνεδρία είναι της μορφής:

rtp://address:port[:ssrc]/content-type/[ttl]

Ο Player δημιουργείται και συνδέεται στο πρώτο stream της συνεδρίας.

Αν υπάρχουν πολλαπλά stream σε μια συνεδρία που θέλουμε να παρουσιάσουμε πρέπει να χρησιμοποιήσουμε τον session manager. Μπορούμε να λαμβάνουμε ειδοποιήσεις από ένα session manager όποτε ένα stream προστίθεται στην συνεδρία και να κατασκευάσουμε ένα νέο Player για κάθε stream. Η χρήση του session manager μας επιτρέπει ακόμη να ελέγχουμε απευθείας και να παρατηρούμε τη συνεδρία.

Transmission

Ένας session manager μπορεί ακόμη να ξεκινάει και να ελέγχει μια συνεδρία μέσα σε ένα δίκτυο. Τα δεδομένα που γίνονται stream χειρίζονται από ένα Processor.

Για παράδειγμα για να δημιουργήσουμε ένα stream αποστολής και να μεταδώσουμε δεδομένα από μια ζωντανή πηγή πρέπει:

1. Δημιουργούμε , προετοιμάζουμε και ξεκινάμε τον SessionManager για τη συνεδρία.
2. Δημιουργούμε ένα Processor χρησιμοποιώντας την κατάλληλη DataSource.
3. Διαλέγουμε το Format αποστολής του Processor. Ένας κατάλληλος RTP ακωδικοποίητης πρέπει να είναι διαθέσιμος για τα δεδομένα που θέλουμε να μεταδώσουμε.
4. Να πάρουμε το DataSource αποστολής από τον Processor.
5. Να καλέσουμε την createSendStream του session manager και να περάσουμε την DataSource.

Μπορούμε να ελέγχουμε την μετάδοση μέσω της SendStream start και stop μεθόδων.

Όταν ξεκινάει ο SessionManager συμπεριφέρεται ως δέκτης. Μόλις χρησιμοποιηθεί η SendStream ξεκινάει να στέλνει RTCP αναφορές αποστολέα και συμπεριφέρεται ως πηγή αποστολής ωσότου ένα η περισσότερα stream υπάρχουν. Αν όλα τα SendStreams έχουν κλείσει (όχι απλά σταματήσει) ο session manager ξανά πηγαίνει σε κατάσταση παθητικού δέκτη.

Extensibility

Όπως όλο το πακέτο JMF έτσι και οι δυνατότητες του RTP έχουν και επιπλέον δυνατότητες όπως:

- Τοποθέτηση ενός πολυπλέκτη ή/και αποπολυπλέκτη.
- Προσθήκη άλλων κωδικοποιήσεων ή/και ειδικών εφέ.
- Χρήση δικών μας πηγών δεδομένων (FTP)
- Χρήση γραμμής χρόνου.
- Δημιουργία αρχείου που αποθηκεύονται τα δεδομένα της συνεδρίας ήχου/βίντεο.

Βιβλιογραφία – Αναφορές

1. Sun's Developer Network (SDN): JMF API Guide: (http://java.sun.com/products/java-media/jmf/)
2. Sun JavaSe: (http://java.sun.com/javase/)
3. Sun Support: (http://java.sun.com/javase/support/)