La Sapienza
Università di Roma

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Anno Accedemico 2008-2009

Corso di Progetto di Reti di Calcolatori e Sistemi Informatici

Prof. Stefano Millozzi

Progetto per un Sistema di Video/Audio Sorveglianza Remota

RemoteGuard

di Pier Paolo Ciarravano

Matr. 773970

Specifica dei Requisiti dell'applicazione RemoteGuard

Il progetto riguarda lo sviluppo di un sistema di video e audio sorveglianza remoto.

Il sistema consiste di un server centrale di controllo e coordinamento, di client remoti che si occupano dell' acquisizione audio/video (sensori) e di un client di gestione e monitoraggio che permette di abilitare i singoli client di acquisizione e accedere ai contenuti audio e video.

Il sistema è costituito da tre parti:

1. Client di Monitoring: chiamato "Controller"
   Il client permette di controllare da remoto i singoli sensori; in particolare permette all'utente di conoscere l'elenco di sensori attivi (Guard di seguito descritto), di sceglierne uno e di riprodurre i contenuti audio e video che il client remoto acquisisce.

2. Client di acquisizione: chiamato "Guard"
   Si tratta di una applicazione che si occupa dell'acquisizione dei contenuti audio e video della webcam e del microfono del PC in cui è in esecuzione. Dopo essere stata avviata, l'applicazione si connette al server centrale, *specificando un nome identificativo della Guard* , e resta in attesa di comandi. Quando il Controller attraverso il Server richiede di acquisire dati (audio/video), accede a webcam e microfono del PC e trasmette i dati al Server, il quale poi li ridirige verso il Controller.

3. Server Centrale: chiamato "Server"
   Il server centrale è responsabile di coordinare le attività tra le varie applicazioni remote di acquisizione e il client di monitoraggio (Controller e Guard). In particolare, quando il Controller richiede di accedere ai contenuti audio/video di una particolare Guard, il Server si occupa di inoltrare i relativi comandi dal Controller alle Guard e di redirigere lo streaming audio/video dalla Guard attiva al Controller.

Linguaggio e ambiente di sviluppo e di esecuzione

- Il progetto è stato totalmente sviluppato in linguaggio Java SE 6 (JDK 1.6) o superiori. La versione di Java SE 6 è stata scelta perchè implementa l'accesso alla componente visuale System Tray del sistema operativo su cui è in esecuzione la JVM. L'interfaccia grafica e l'interazione con le diverse componenti del progetto utilizzano un menu di accesso che viene visualizzato nella System Tray di sistema.
- Si è utilizzata la IDE di sviluppo Eclipse 3.2.1 "Galileo" per ogni fase del processo di sviluppo.
- Tutte le componenti sviluppate sono eseguite dalla JVM in modalità "standalone" e pertanto non è necessario nessun particolare server di esecuzione o DBMS di supporto (Tomcat, Mysql, ecc.).
- E' stata implementata sia l'acquisizione video che quella audio.
- Requisito necessario per l'esecuzione di tutte le componenti del progetto è la presenza della libreria Sun JMF (Java Media Framework) API vers.2.1.1e correttamente installata nel sistema. Le diverse componenti utilizzano questa libreria sia per l'accesso alla webcam e al sistema di acquisizione audio sia per la comunicazione e lo streaming RTP (Real-time Transport Protocol).
- E' stata utilizzata la libreria Apache Log4j 1.2 per la gestione di tutti i messaggi di log delle componenti dell'applicazione.
- E' stata utilizzata la libreria open source Simple Reliable UDP per la gestione della comunicazione affidabile dei pacchetti UDP per il protocollo di controllo tra le componenti dell'applicazione.

Protocollo di comunicazione utilizzato

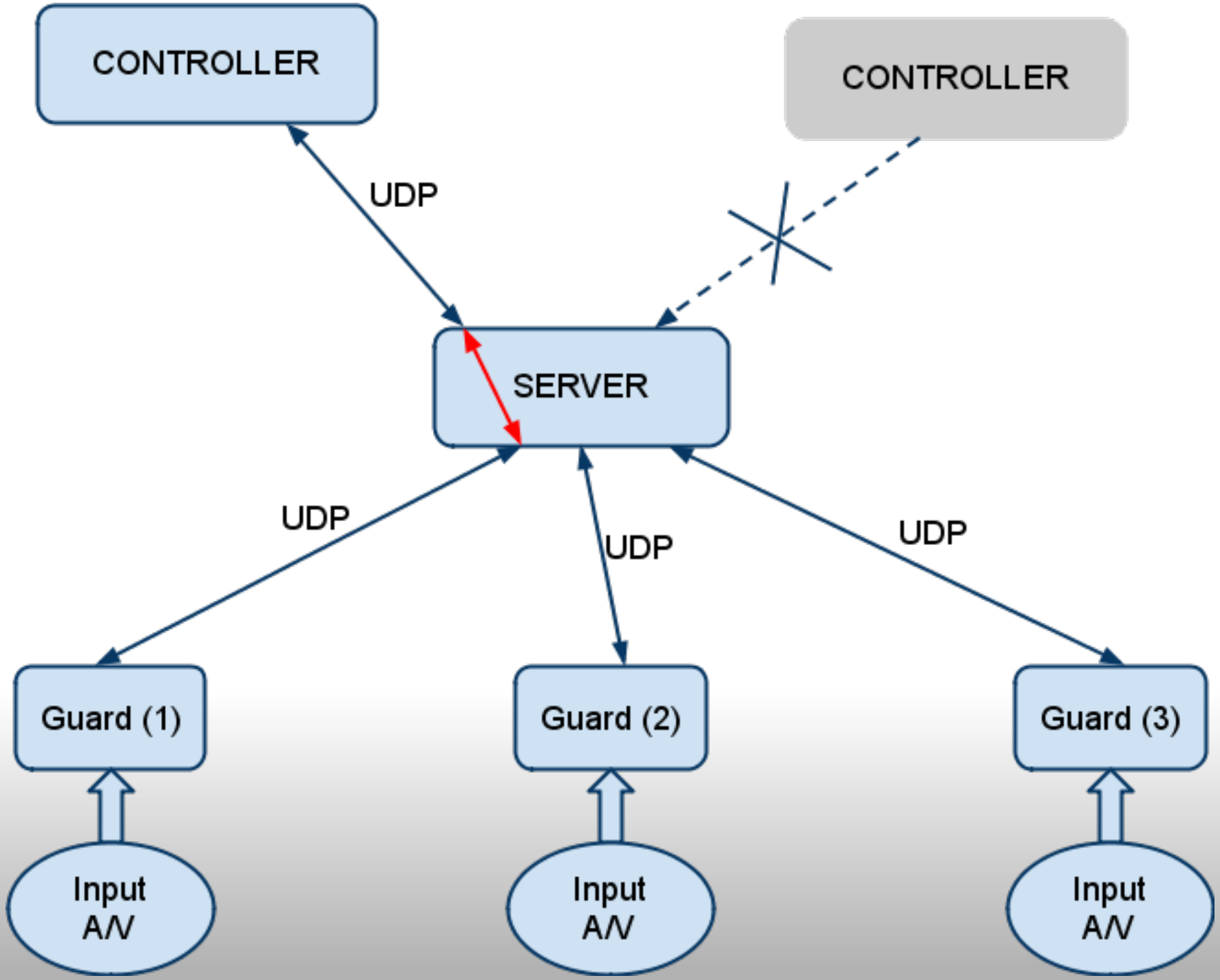
Si è scelto di implementare tutta la comunicazione tra le componenti dell'applicazione (Server/Guard/Controller) utilizzando il protocollo UDP. Tale scelta è giustificata dalle specifiche che prevedono uno streaming di dati audio/video in tempo reale: il protocollo TCP non è indicato per questo tipo di comunicazione in quanto ha un alto overhead.

Il protocollo UDP è un protocollo molto leggero, ma anche se ha un minore overhead rispetto al TCP, è un protocollo senza connessione e non garantisce né che i pacchetti arrivino a destinazione né che i pacchetti arrivino con l'ordine nel quale sono stati inviati; pertanto i sistemi ricevente e trasmittente si devono fare carico a livello applicativo di controllare e compensare questa eventuale perdita di informazione.

Tutta la comunicazione delle componenti dell'applicazione Guard e Controller avviene su un'unica porta UDP in ascolto sul sistema su cui è in esecuzione l'applicazione Server.
A livello applicativo, come vedremo in seguito, è stato implementato un multiplex dei messaggi/comandi di controllo custom per coordinare le attività delle applicazioni, dello streaming audio e video e dei canali di controllo audio/video del protocollo RTP.

La connessione delle Guard e del Controller con il Server avviene specificando 2 diverse password, una per il Controller e una per le Guard. Il Server tuttavia non conosce queste password ma solamente la loro hash md5, ed è solo questa che viene inviata tra i clients e il server, questo per implementare una sicurezza "minima" e non permettere a chi accede al sistema dove è presente il server, di poter connettere eventuali Controller per visualizzare le sorgenti audio/video.

Schema dei componenti del progetto



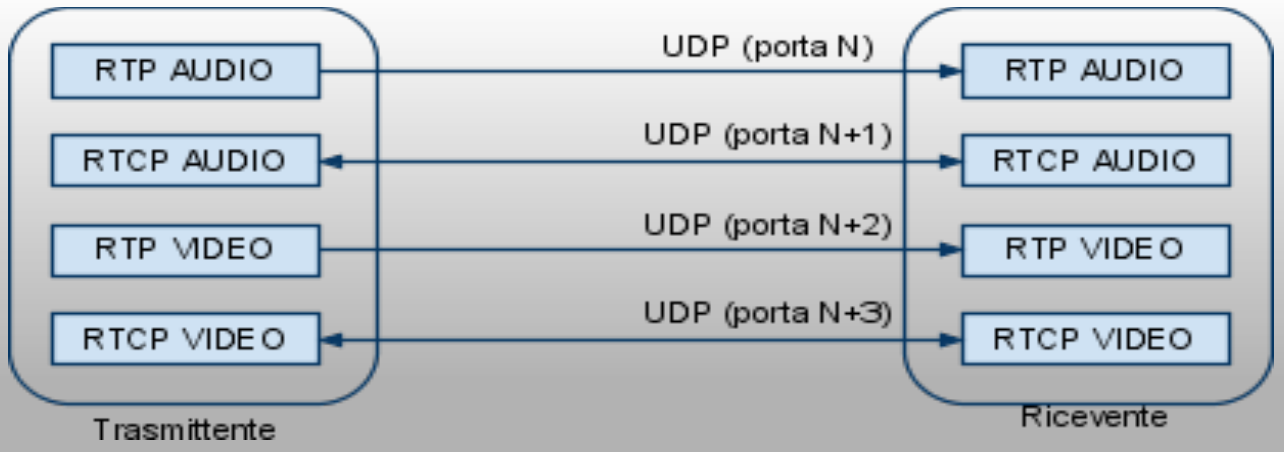
Acquisizione audio e video, JMF e RTP

E' stata delegata tutta la logica per l'acquisizione video delle sorgenti di input video presenti nel sistema (webcam, schede di acquisizione video, tuner tv) e delle sorgenti audio (microfono, scheda di acquisizione audio) alla libreria Sun JMF (Java Media Framework) API vers.2.1.1e. Tale libreria implementa l'accesso a questi dispositivi di acquisizione con facilità ed inoltre è disponibile per diversi sistemi operativi.

Dato che le stesse JMF API implementano tutta la logica di streaming audio e video tra due o più applicazioni utilizzando il protocollo RTP, si sono utilizzate anche queste API per implementare lo streaming in tempo reale e la visualizzazione video e la riproduzione audio delle sorgenti.

Il protocollo RTP è uno standard Internet per il trasporto di dati multimediali ed è implementato in UDP. Il protocolo RTP non garantisce però la qualità di servizio, pertanto il trasporto e la consegna dei dati sono coadiuvati attraverso un protocollo di controllo RTCP (Real-time Transport Control Protocol); RTP e RTCP sono definiti in modo da essere indipendenti dai sottostanti livelli di trasporto e di rete. JMF mette a disposizione un'interfaccia chiamata javax.media.rtp.RTPConnector la quale permette, implementando i suoi metodi, di poter implementare la comunicazione RTP/RTCP attraverso un protocollo personalizzato.

Una comunicazione RTP standart usata da JMF per l'invio/ricezione di audio e video utilizza 4 porte UDP in ricezione e trasmissione dei pacchetti: 2 porte per i dati audio e video e altre due porte per i corrispondenti canali di controllo RTCP, come descritto dal diagramma di seguito mostrato:



Codifica audio e video di JMF

Per quanto riguarda la codifica di compressione Video da utilizzare con lo streaming RTP, JMF mette a disposizione sia lo standard H.263 che lo standard JPEG Video. Entrambi gli standard sono implementati in maniera trasparente allo sviluppatore che usa le API JMF.

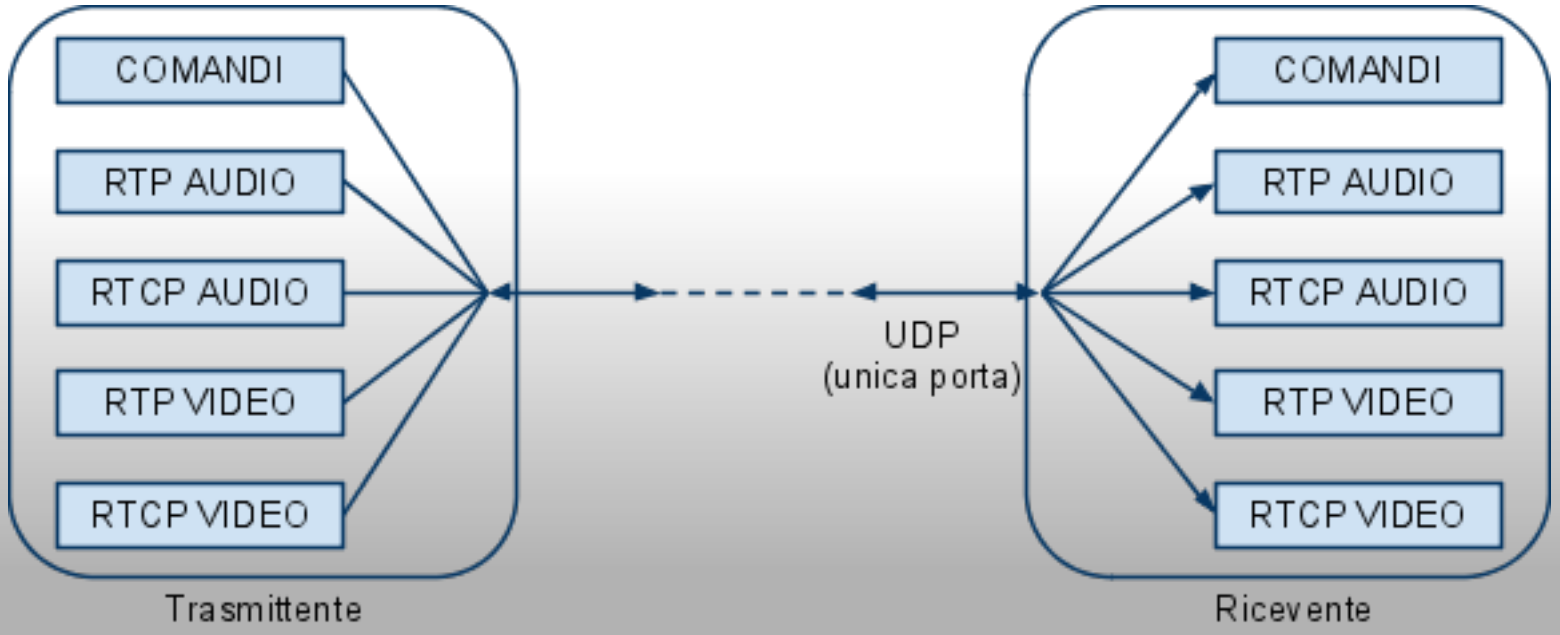
Per la codifica audio su streaming RTP, JMF mette a disposizione numerosi formati con diverse qualità di acquisizione delle sorgenti audio.

Attraverso un file di configurazione di properties testuali all'interno dell'applicazione, è possibile scegliere il formato audio e video desiderato tra quelli disponibili e implementati dalle diverse versione della libreria JMF installata sul sistema.
Inoltre qualora si scelga il formato JPEG per la codifica video, attraverso il file di configurazione si può specificare il livello di compressione della qualità delle immagini.

Per JMF ogni flusso di dati multimediali audio o video rappresenta una sessione RTP; queste sessioni potrebbero essere visualizzate tramite player audio e video standard forniti dalle API JMF, uno per la sessione audio e uno per la sessione video, sincronizzando perfettamente attraverso un'opportuna chiamata alle API le due sorgenti, questo grazie al fatto che ogni pacchetto RTP ha un proprio timestamp. Si è scelto tuttavia di visulizzare i "datasource" (javax.media.protocol.Datasource) audio e video in un unico player standard, facendo un "merge" *(javax.media.Manager.createMergingDataSource)* delle due sessioni. Allo stato attuale dell'applicazione, utilizzando questa procedura non è stato però ancora possibile la sincronizzazione in maniera perfetta delle sorgenti audio e video.

Multiplexing dei dati audio/video e comandi di controllo

La principalmente attenzione, oltre che a sviluppare tutte le specifiche richieste dal documento di progetto, si è focalizzata nell'implementare tutta la comunicazione di rete utilizzando esclusivamente il protocollo UDP. Oltre alle 4 porte UDP necessarie per la comunicazione audio/video su protocollo RTP, si avrebbe la necessità di un'ulteriore porta per i pacchetti di controllo e comando applicativo tra le varie componenti Guard/Controller/Server.
Si è pertanto scelto di effettuare un multiplex di tutte le 5 porte UDP, utilizzando un pacchetto personalizzato (ppc.remoteguard.UDPPacket) che contiene oltre ai dati del pacchetto UDP che si vuole inviare, il tipo di pachetto tra i 5 possibili e l'id assegnato a ciascun client (Guard o Controller) che si connette con il Server. Inoltre è stata creato una gestore di code di messaggi (ppc.remoteguard.UDPPacketQueue), che fornisce e divide i messaggi in base al client da cui si è ricevuto il pacchetto e il tipo di pacchetto ricevuto (tra i 5 possibili). In questo modo tutta la comunicazione tra il server centrale e il client di monitoring e di acquisizione si svolge sulla stessa porta UDP.
Tramite l'implementazione dell'interfaccia javax.media.rtp.RTPConnector, attraverso le classi: ppc.remoteguard.UDPMultiplexerClient e ppc.remoteguard.UDPMultiplexerServer è stato poi possibile effettuare il demultiplex dei pacchetti audio e video per ricreare i 2 streaming RCP/RTCP.



Protocollo di controllo: Reliable UDP

Per permettere alle diverse componenti dell'applicazione (Server, Guard e Controller) di coordinare i compiti richiesti, si è sviluppata una classe (ppc.remoteguard.CommandMessage) che rappresenta e contiene i vari comandi da compiere (descritti nelle slides successive) con i relativi parametri di controllo possibili per ogni comando. Le istanze di questa classe sono successivamente letti e scritti attraverso ObjectInputStream e ObjectOutputStream.

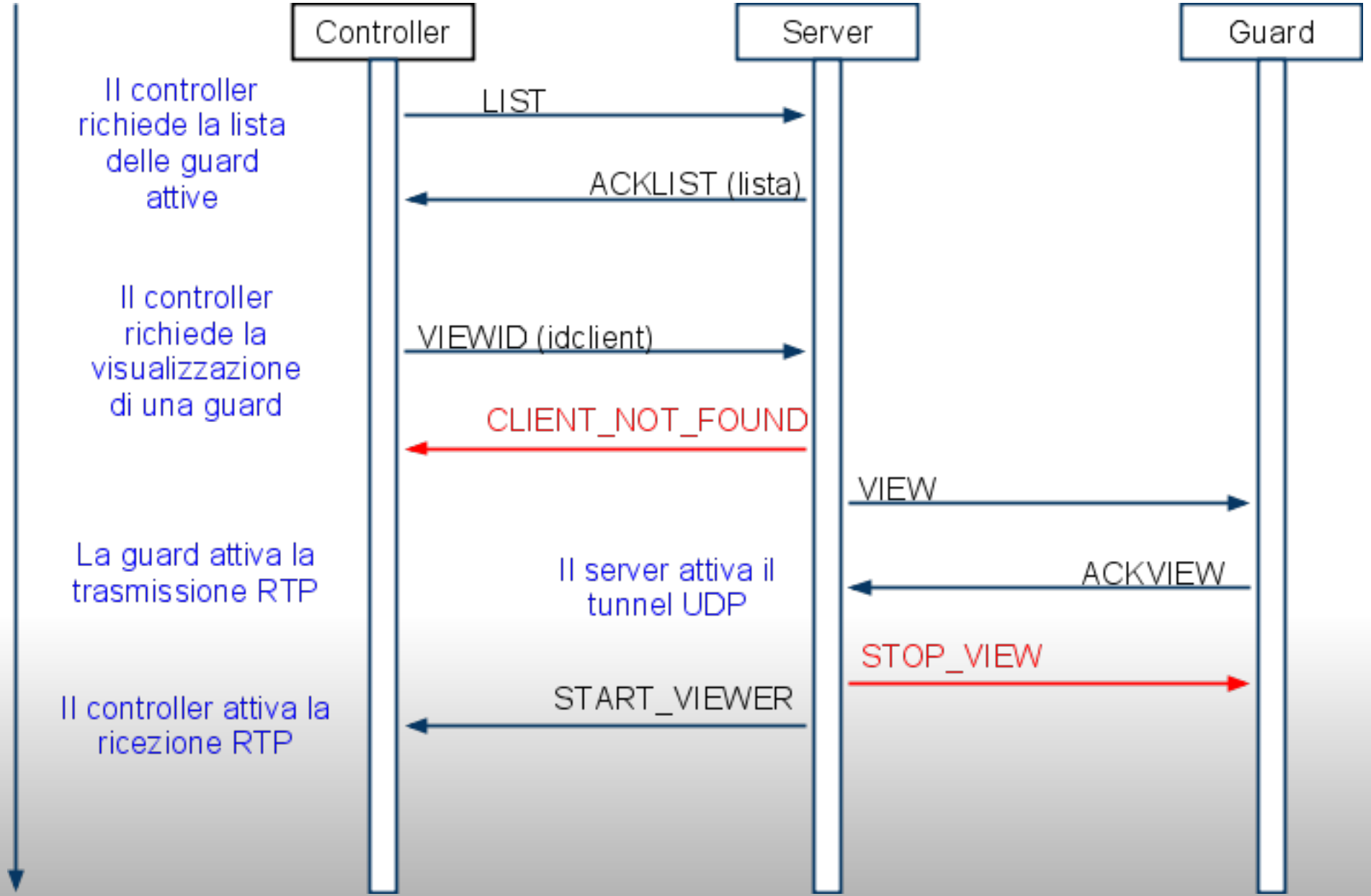
Mentre per lo streaming audio/video, i protocolli RTP e RTCP si occupano completamente della qualità di servizio, per tutti i messaggi di controllo inviati in UDP si avrebbe il problema di implementare a livello applicativo la logica di controllo per compensare l'eventuale perdita di informazione insita nel protocollo UDP. Si è pertanto scelto di implementare solo per i pacchetti (UDPPacket) di controllo il protocollo basato sull'Internet Draft (del Febbraio 1999) degli autori Bova, Krivoruchka e Cisco Systems (1999) intitolato "Reliable UDP" (http://www.ietf.org/old/2009/proceedings/99mar/I-D/draft-ietf-sigtran-reliable-udp-00.txt); Tale protocollo implementa il controllo dell'effettiva ricezione dei pacchetti nell'ordine in cui sono stati inviati e pertanto permette la comunicazione affidabile a livello applicativo sul protocollo UDP in modo totalmente trasparente allo sviluppatore.

L'implementazione del protocollo "Reliable UDP" è affidata alla libreria open source "Simple Reliable UDP" (http://rudp.sourceforge.net/). Questa libreria crea, utilizzando un java.net.DatagramSocket, un InputStream e un OutputStream, completamente affidabili, come se fossero due stream di una connessione TCP. La comunicazione delle instanze di ppc.remoteguard.CommandMessage avviene utilizzando questi streams.

Per rendere affidabili solamente i pacchetti di controllo e non i pacchetti di tipo RTP/RTCP si è estesa la classe java.net.DatagramSocket (da passare alla libreria "Simple Reliable UDP" che la utilizza per la ricezione e l'invio UDP) tramite la classe ppc.remoteguard.rudp.DatagramWrapper, implementando i metodi send e receive, che utilizzano le classi ppc.remoteguard.UDPMultiplexerClient e ppc.remoteguard.UDPMultiplexerServer per l'invio personalizzato dei pacchetti.

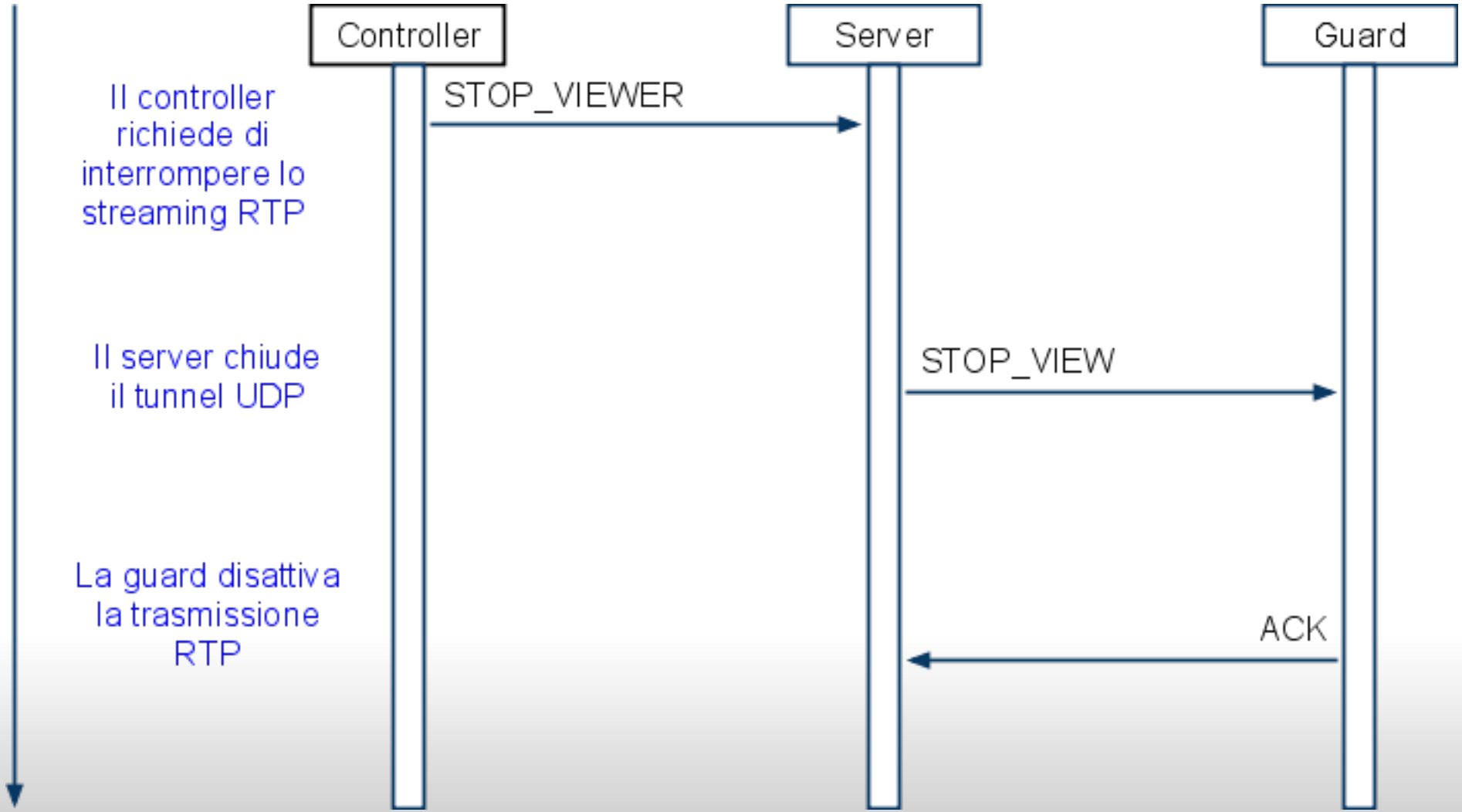
Descrizione dei comandi del protocollo di controllo

Sequence Diagram che descrive lo scambio dei messaggi per la scelta, l'attivazione e la visualizzazione di una Guard



Descrizione dei comandi del protocollo di controllo

Sequence Diagram che descrive la richiesta di interruzione dal controller dello streaming RTP



Struttura dei packages delle classi

Tutte le classi dell'applicazione sono nel package ppc.remoteguard che comprende i seguenti sotto packages:

- ppc.remoteguard : classi comuni a tutte le componenti main dell'applicazione (Server, Guard e Controller)
- ppc.remoteguard.controller : classi del main del client di monitoraggio
- ppc.remoteguard.guard : classi del main del client di acquisizione
- ppc.remoteguard.server : classi del main del server centrale
- ppc.remoteguard.log : classi per la gestione del log con Log4j
- ppc.remoteguard.resources : icone gif utilizzate dal SystemTray e dalle finestre
- ppc.remoteguard.rudp : wrapper per la gestione del canale Reliable UDP
- ppc.remoteguard.util : classi di utilita' varie

Per le tre componenti del progetto (Server, Guard e Controller) sono state sviluppate le seguenti classi main che ne implementano la logica di controllo:

- ppc.remoteguard.server.Server: Main per l'applicazione server centrale
- ppc.remoteguard.guard.Guard: Main per il client di acquisizione
- ppc.remoteguard.controller.Controller: Main per il client di Monitoring

Scelte GUI di input/output e file di configurazione

L'interazione grafica delle tre componenti del progetto è abbastanza semplice in quanto non prevede particolari e complesse interfacce. Dato inoltre che si tratta di applicazioni che (salvo il Controller) non richiedono particolari interazioni durante la loro esecuzione (Server e Guard) si è scelto di utilizzare un semplice menu visualizzabile con un'icona che compare nella SystemTray applicativa. L'accesso a questa componente avviene utilizzando la classe java.awt.SystemTray presente dalle versioni di Java 6 SE e successive.
Pertanto tutte le possibili operazioni vengono lanciate attraverso il menu contestuale (accessibile tramite il tasto destro del mouse) sull'icona relativa all'applicazione lanciata nella SystemTray.
Inoltre sono stati utilizzati messaggi di pop-up di sistema sulla System Tray, per segnalare eventuali semplici eventi, come la connessione avvenuta con successo di una Guard al server, o come l'inizializzazione corretta del Server.

E' stata creata una finestra per visualizzare il log dell'applicazione in output tramite le API di Apache Log4j.

La visualizzazione dello straming audio/video avviente tramite una semplice finestra che contiene un controllo standard JMF per lo streaming audio/video.

Le interfacce dei parametri di configurazione della connessione e della scelta della Guard da visualizzare per il Controller sono semplici interfacce grafiche di input tramite JTextField e JComboBox, con controllo e validazione dei dati inseriti. Alcuni alert dell'applicazione sono stati sviluppati utilizzando chiamate ad JOptionPane.showMessageDialog.

Oltre al file di configurazione di log4j (log4j.properties) che specifica i diversi formati e il livello di log desiderato (INFO, DEBUG,...), e' presente un file di configurazione, al livello della root classpath, chiamato constant.properties che dettaglia tutti i parametri di configurazione dell'applicazione che non richiedono particolari cambiamenti in stato di esecuzione, e i possibili parametri di default che compaiono nelle finestre di configurazione, come ad esempio la porta UDP da utilizzare per effettuare la connessione.

Miglioramenti futuri

- Perfetta sincronizzazione audio/video
- Minore accoppiamento delle classi