UNIVERSITÀ DEGLI STUDI DI BOLOGNA

FACOLTÀ DI INGEGNERIA

Corso di Laurea in Ingegneria Informatica

Reti di Calcolatori

Adattamento di servizi multimediali per sistemi pervasivi

Tesi di laurea di:

Massimo Manarini

Relatore

Chiar.mo Prof. Ing. Antonio Corradi

Correlatore

Ing. Luca Foschini

Anno Accademico 2004 - 2005

Indice

Parole chiave: **Servizi multimediali**
Qualità del servizio
Java Media Framework
Sistemi basati su proxy
Erogazione multipercorso

INTRODUZIONE 5

CAP. 1 CARATTERISTICHE SERVIZI MULTIMEDIALI E QUALITÀ DEL SERVIZIO 7

1.1 CARATTERISTICHE DEI FLUSSI MULTIMEDIALI 8
1.1.1 Larghezza di Banda 8
1.1.2 La latenza 8
1.1.3 Il jitter 9
1.1.4 Lo skew 10
1.2 LA NECESSARIETÀ DELLE RISORSE 10
1.3 PROTOCOLLO TCP/IP E QUALITÀ DEL SERVIZIO 11
1.4 ACCORGIMENTI PER LA QUALITÀ DEL SERVIZIO 13
1.4.1 Strategia a XOR 14
1.4.2 Strategia con doppia codifica 15
1.4.3 Interleaving 16
1.4.4 Utilizzo dei percorsi alternativi 16
1.5 RTP 17
1.5.1 Caratteristiche del protocollo RTP 17
1.6 RTCP 18
1.6.1 Caratteristiche del protocollo 18
1.7 CONCLUSIONE 19

CAP. 2 JAVA MEDIA FRAMEWORK: ANALISI DELLA LIBRERIA E DEI SUOI LIMITI 20

2.1 UNO SGUARDO GENERALE 20

Indice

2.2 ARCHITETTURA DEL JMF 20
2.2.1 Modello ad eventi del JMF *24*
2.2.2 Modello dei dati *25*
2.2.3 Formato dei dati *26*
2.2.4 Controlli del JMF *26*
2.3 PRESENTAZIONE DEI FLUSSI MULTIMEDIALI. 28
2.3.1 Player *29*
2.3.2 Processor *30*
2.3.3 DataSink *31*
2.4 TRASMISSIONE DEI FLUSSI MULTIMEDIALI ATTRAVERSO LA RETE 31
2.4.1 RTP *31*
2.4.1.1 Architettura RTP nel JMF *33*
2.4.1.2 Session Manager *33*
2.4.1.3 Eventi Rtp *34*
2.6 APPROFONDIMENTI DEL JMF PER IL PROGETTO 36
2.6.1 DataSource *36*
2.6.2 Player *37*
2.6.2.1 Dettagli sulla creazione del Player 37
2.6.3 Processor *38*
2.6.3.1 Dettagli sulla creazione del Processor 38

CAP. 3 MODIFICHE DEL JMF 39

3.1 IL PACKAGE COM.SUN.MEDIA 39
3.2 LIVELLI DI MODIFICA 39
3.3 APPROFONDIMENTO SULLE MODIFICHE 40
3.3.1 MediaPlayer *41*
3.3.2 MediaProcessor *42*
3.3.3 PlayBackEngine *43*
3.3.4 ProcessEngine *44*
3.3.5 BasicSourceModule *44*
3.3.6 RawBufferParser *45*

CAP. 4 PROGETTO 47

4.1 STRUMENTI UTILIZZATI 48

4.1.1 Creazione dei video utilizzati per i test *48*
4.2 SCENARI DI STUDIO 49
4.3 LIBRERIA DI TEST PER IL MONITORAGGIO DELLE RISORSE (PACKAGE PROGETTO.TEST). 50
4.3.1 Soluzione con il JMF standard *50*
4.3.2 Soluzione con il JMF modificato *52*
4.4 UTILIZZO DELLA LIBRERIA PER IL TEST 52
4.4.1 Dettagli libreria test *53*
4.5 APPLICAZIONE DI PRESENTAZIONE (PACKAGE PROGETTO.PRESENTAZIONE) 55
4.5.1 Approfondimento delle Api *55*
4.5.1.1 ServerController e ProxyController 56
4.5.1.2 Server e Proxy 57
4.5.1.4 Dettagli funzionamento Proxy 57
4.5.1.5 Dettagli funzionamento Server 57
4.5.1.3 GuiServer e GuiProxy 58

CAP. 5 ANALISI TEST E PRESENTAZIONE APPLICAZIONE 59

5.1 TEST E VALUTAZIONE DEI RISULTATI 59
5.1.1 Macchine su cui sono stati effettuati i test *60*
5.1.2 Risultati ottenuti *61*
5.1.3 Riassunto test *62*
5.1.4 Considerazioni sui risultati *63*
5.2 PRESENTAZIONE APPLICAZIONE 64
5.2.1 Strumenti utilizzati *64*
5.2.2 Fasi della simulazione *64*
5.2.2.1 Dettagli della simulazione 64

CONCLUSIONI 66

RIFERIMENTI BIBLIOGRAFICI 67

Introduzione

Dai primi anni novanta ad oggi, abbiamo assistito ad un largo ed imponente sviluppo di Internet e contestualmente alla nascita e allo sviluppo di nuovi servizi ad esso collegati. Accanto ai servizi tradizionali, e.g., web e posta elettronica, gli utenti richiedono servizi sempre più avanzati, e.g., servizi per l'erogazione di materiale multimediale come radio o video broadcasting, fino ad arrivare all' integrazione di Internet con la rete telefonica tradizionale e dei relativi servizi offerti da questi due mondi, e.g., web sui cellulari oppure servizi di telefonia sopra IP (Voice over IP, VoIP).

La corretta fornitura di servizi multimediali di ultima generazione richiede però garanzie assai stringenti sulla qualità di servizio offerta che non erano state previste nel progetto iniziale di Internet. Per far fronte a tali carenze sono state proposte diverse infrastrutture distribuite di supporto volte a migliorare la qualità dei servizi multimediali offerti. In particolare, recenti proposte architetturali suggeriscono il coinvolgimento nell'erogazione del servizio non solo del fornitore e del fruitore, i.e. servitore e cliente, ma anche di altre entità intermedie, i.e. proxy, che possono operare adattamenti o modifiche sui flussi multimediali erogati con lo scopo di migliorare la qualità del servizio percepita dall' utente finale.

La realizzazione dei servizi multimediali e delle infrastrutture di supporto richiede però un notevole sforzo in termini sia di tempi di sviluppo sia di gestione dell'infrastruttura distribuita. Con lo scopo di facilitare il progetto di tali servizi e ridurne il tempo di sviluppo sono quindi stati sviluppati, nei più comuni linguaggi di programmazione, diversi framework multimediali.

Questa tesi nasce con l'obiettivo di migliorare il Java Media Framework (JMF), un framework scritto in Java per la costruzione di software atti all'erogazione di servizi multimediali. In particolare, si vogliono estendere le Application Program Interface (API) offerte da JMF in modo da supportare la fornitura di servizi multimediali con un sistema multi-percorso (multi-path). Lo scenario applicativo prevede che ci siano alcuni server che erogano uno stesso flusso multimediale, un proxy e un client. Si vuole modificare il JMF, in modo da poter cambiare sul proxy la sorgente dati (server) dalla quale viene erogato il flusso multimediale, senza dover in alcun modo modificare il percorso di servizio dal proxy al client (si vuole cioè rendere tale operazione completamente trasparante al client). Le nuove funzionalità introdotte saranno poi

utilizzate per realizzare lo scenario applicativo proposto e una fase estensiva di testing concluderà il lavoro di tesi.

La tesi è organizzata come segue. Nel primo capitolo vengono introdotte le caratteristiche dei servizi multimediali e alcune definizioni legate alla qualità del servizio con un attenzione particolare sui parametri che la valutano e la modificano; la presentazione del protocollo RTP e RTCP conclude il capitolo. Nel secondo capitolo si presenta il JMF e si discutono i limiti del framework per quanto concerne la creazione di un architettura con sistema a multipath. Nel terzo capitolo vengono mostrate e spiegate le modifiche fatte al JMF per aggiungere la nuova funzionalità. Il quarto capitolo mostra il funzionamento della libreria e presenta l'applicazione realizzata per testarne il funzionamento. Il quinto e ultimo capitolo conclude la tesi mostrando i risultati dei test effettuati con la libreria e con l'applicazione sviluppata.

Cap. 1 Caratteristiche servizi multimediali e qualità del servizio

I servizi multimediali, a differenza dei normali servizi di gestione o trasporto dati, hanno delle caratteristiche meno rigorose in termini di qualità dei dati ma stringenti per altri vincoli che se non rispettati possono pregiudicare seriamente il servizio stesso; questi vincoli sono il tempo e la (quantità di banda del servizio stesso).

Per loro natura i flussi multimediali, quali audio o video, non necessitano di assoluta perfezione nella qualità del segnale; se l'immagine è leggermente distorta o se un suono è modificato entro certi limiti che definiremo meglio in seguito, non viene pregiudicato il servizio. Se al contrario arrivano dei dati in ritardo o se la banda non è sufficiente a trasportare tutti i dati in tempo reale il servizio può risultare seriamente compromesso.

Molti anni fa, i servizi di trasmissione audio e video via etere (radio e tv che sono i più utilizzati ancora oggi in molte parti del mondo) avevano problemi di affidabilità del segnale e di quantità di banda del segnale.

Con l'avvento delle tecnologie digitali, sono nati diversi tipi di problemi.

Per utilizzare tali tecnologie, tutti i segnali analogici devono essere convertiti in digitale elborati e riconvertiti; sono nati allora problemi relativi alla dimensione dei dati: un segnale audio o video di buona qualità può generare una quantità di dati enorme.

Un segnale audio della durata di 3 minuti ad una qualità di 16 bit per campione con 44100 campioni al secondo genera più di 120 Mbit di dati.

Un filmato non compresso con qualità video TV standard occupa all'incirca 120 Mbit al secondo.

Se si pensa alla capacità di un modem di trasportare all'incirca 48-56kbit al secondo si può ben comprendere come questo non sia un problema da poco.

Se a questo aggiungiamo, per natura strutturale della rete internet, dei ritardi dei pacchetti dalla sorgente alla destinazione, vediamo necessario che siano rispettate determinate caratteristiche; pena l'impossibilità di usufruire del servizio.

Nei paragrafi successivi discuteremo delle caratteristiche dei flussi multimediali e della necessarietà delle risorse. Valuteremo quindi i limiti del protocollo TCP/IP per questo genere di servizi e concluderemo con il protocollo RTP e le considerazioni sulla qualità del servizio.

Cap. 1 Caratteristiche servizi multimediali e qualità del servizio

1.1 Caratteristiche dei flussi multimediali

Per poter comprendere appieno le problematiche legate ai servizi multimediali è necessario spiegare le principali caratteristiche dei flussi multimediali.

1.1.1 Larghezza di Banda

La larghezza di banda è la caratteristicha che indica la portata di un mezzo di trasmissione digitale; stabilisce la quantita di dati che possono transitare nell'unità di tempo. L'unita di misura per questa grandezza è solitamente espressa in bit/secondo (bps) o byte/secondo e risulta una caratteristica molto limitativa per i servizi multimediali. Spesso e volentieri infatti i servizi richiedono una banda molto ampia riducibile solamente con l'utilizzo di opportuni algoritmi di compressione a scapito delle risorse del sistema; l'attività di compressione e decompressione richiede notevoli risorse di memoria e potenza di calcolo.

1.1.2 La latenza

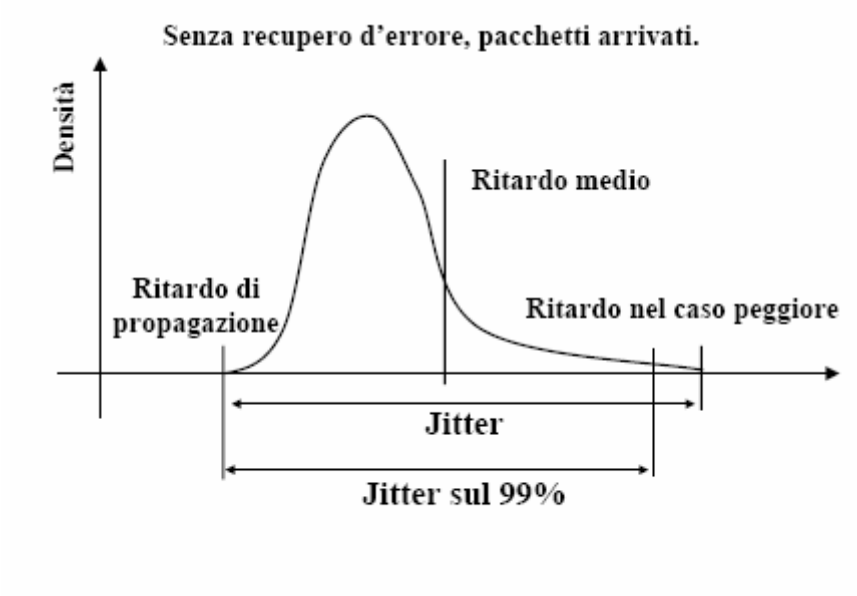
La latenza è il parametro che misura il ritardo introdotto dal mezzo di trasmissione, ovvero il tempo impiegato dai dati per transitare dalla sorgente al destinazione. Nei servizi multimediali questo parametro è forse il più importante perché in grado di far degradare il servizio, se vengono oltrepassati dei limiti stabiliti.

Per servizi di videoconferenza o di controllo remoto, è stato valutato un ritardo massimo tollerabile pari a 150 ms. Naturalmente per caratteristiche fisiche dei mezzi di trasmissione digitale come la rete internet, questa caratteristica non è mai costante, ma è soggetta a variazioni continue e sarà quindi rappresentata da una variabile aleatoria. Questo fenomeno aleatorio ci obbliga a prendere in considerazione un'altra caratteristica direttamente collegata: il jitter.

1.1.3 Il jitter

Il jitter ci informa sulla varianza della latenza, in altre parole lo scostamento di questa dal suo valore medio (Figura 1.1). Uno scenario perfetto vedrebbe i pacchetti giungere al ricevente nello stesso ordine nel quale sono stati generati, e processati con la stessa velocità con la quale arrivano.
Nella realtà questo scenario è impossibile; perciò è necessario utilizzare dei buffer per sincronizzare e riordinare i dati che arrivano al ricevente.
La dimensione dei buffer è direttamente collegata con il jitter.
Una gran variazione della latenza dalla media rende necessaria una grande dimensione dei buffer; questo ci permette di valutare lo stato del carico di lavoro del mezzo trasmissivo.

Figura 1.1: Latenza dei pacchetti



Un'altra caratteristica importante per descrivere un flusso multimediale è il *loss rate* che quantifica la percentuale di dati che un flusso è in grado di perdere senza deteriorarsi in maniera sensibile. Tale parametro è ortogonale agli altri, poiché possono esistere flussi diversi aventi ugual jitter ed uguale occupazione di banda, ma con loss rate molto diverso a causa del tipo di dati trasmessi.

1.1.4 Lo skew

Lo skew è un parametro che indica la bontà riguardante la sincronizzazione tra due flussi multimediali che partono dal medesimo mittente e devono giungere alla stessa destinazione .
Questo calcolo si effettua come differenza tra le latenze (medie) dei due flussi; è molto importante che sia il più basso possibile in quanto un suo alto valore porta a sgradevoli situazioni. Ci sono tantissimi servizi multimediali che gestiscono più flussi multimediali (video conferenza per esempio); audio e video vengono spesso spediti in canali separati.

1.2 La necessarietà delle risorse

Le applicazioni multimediali per loro natura necessitano di molte risorse; questa è la principale causa che rallenta il loro sviluppo.
In particolare le risorse più significative sono le seguenti: la CPU, la banda del mezzo trasmissivo e la memoria.
La CPU è la risorsa più contesa perché viene coinvolta in diverse fasi di gestione e della presentazione del servizio multimediale:

- impacchettamento/spacchettamento dei dati;
- codificazione/decodificazione del flusso (cioè l'esecuzione di algoritmi di compressione/decompressione);
- rendering dell'oggetto multimediale.

Queste sono tra le operazioni più costose in termini di risorsa CPU. Solitamente i requisiti, per un dato tipo di CPU, vengono espressi come frazioni di tempo di un periodo, oppure sotto forma di percentuale dei cicli del microprocessore necessari.
La banda trasmissiva è un'altra risorsa solitamente richiesta in grandi quantità.
Non dimentichiamo la richiesta di banda di un filmato non compresso con qualità video TV standard che si aggira all'incirca sui 120 Mbps (Mega bit per secondo), eccedendo la capacità di molte connessioni Ethernet pari a 100 Mbps, oggi utilizzate nella maggior parte degli ambienti aziendali. Questo problema può essere ridotto notevolmente grazie alla compressione dei dati spostando l'utilizzo di risorse dalla banda trasmissiva alla

risorsa CPU. Utilizzando la prima delle codifiche studiate dal Moving Picture Experts Group (MPEG), cioè MPEG-1, possiamo trasmettere un video consumando solamente 1,5 Mbps di banda.

L'ultima risorsa che consideriamo in questa carrellata di caratteristiche dei servizi multimediali è la memoria. I flussi multimediali richiedono molta memoria e cosi anche la loro manipolazione (gestione compressione e decompressione); ogni fase di lavorazione di un flusso multimediale richiede immagazzinamento in appositi buffer occupando memoria. La robustezza e la resistenza ad errori nei servizi multimediali rende necessario un aumento vertiginoso della memoria per tenere più informazioni del flusso. Questa importante risorsa, avendo un costo rispetto alle altre molto basso ed essendo di conseguenza presente sempre più spesso in grandi quantità all'interno dei dispositivi, produce meno problemi arrivando difficilmente alla saturazione.

1.3 Protocollo TCP/IP e Qualità del servizio

Internet è la rete telematica che collega tutto il mondo. Al suo concepimento, aveva scopi di utilizzo molto diversi da quelli attuali. L'obbiettivo principale dei fautori era creare una rete resistente e affidabile in grado di portare le informazioni a destinazione in ogni caso; la chiave di progettazione era la "migliore efficienza" possibile.

Fisicamente la rete è costruita come una ragnatela che tramite svariati nodi collega i computer di tutto il mondo. Si è ingrandita mantenendo la stessa struttura e aumentando di giorno in giorno il numero di sottoreti collegate.

I nodi hanno il compito di far transitare i dati da una rete all'altra e, tramite una serie di tecniche ingegneristiche, scelgono la strada "migliore" da far percorrere ai dati per garantire il loro arrivo a destinazione.

I dati viaggiano tramite pacchetti indipendenti; può quindi succere che due pacchetti con la medesima sorgente e il medesimo destinatario effettuino percorsi diversi.

Questo sistema nel suo complesso si è scoperto magnificamente scalabile: agli albori non si pensava ad un collegamento dell'intero globo terrestre e quando lo sviluppo della rete, negli anni 80-90 prese una piega esponenziale, si rimase stupidi per il buon funzionamento della rete.

Questa robustezza e scalabilità è dovuta alla progettazione dello stack di protocolli

legato alla rete comprendente sia la parte hardware che la parte software.
Questo stack è chiamato TCP/IP

La suite di protocolli offerti per la trasmissione dei dati non può per natura strutturale dare garanzie sulle tempistiche di consegna dei dati e dei ritardi degli stessi.
Le pagine web lavorano con il protocollo HTTP basato su TCP e ogni interazione, tra il Client e il Server, è indipendente. La rete si può congestionare all'improvviso, e vista la velocità con cui nuovi utenti entrano in rete ogni giorno e con le molteplici situazioni anomale dovute a spam, virus e altro, è normale considerare che una connessione dapprima veloce e affidabile possa in breve tempo diventare una connessione lenta e inaffidabile.

Fino a pochi anni fa, i pochi servizi multimediali esistenti, presenti nelle pagine web come video o audio, funzionavano nel modo seguente:

- Il Client si connetteva al Server e scaricava la pagina web
- Il contenuto multimediale, a seconda se la pagina lo richiedeva alla fine del caricamento o con un click su un link presente all'interno, veniva scaricato tutto e solo alla fine veniva presentato.

Questo sistema potrebbe funzionare per i servizi multimediali futuri se venisse garantita sempre e comunque una banda sufficiente a non creare congestioni tra i nodi della rete.
Per come è costruita oggi la rete internet, è impossibile garantire ciò. Ed è soprattutto impossibile garantire uno streaming/broadcast continuo di informazioni.

Passiamo quindi a valutare i protocolli attuali per poter capire qual'è la migliore strada da percorrere per affrontare i problemi legati all'erogazione dei servizi multimediali.
Lavorando a livello software analizzeremo i livelli dello stack dal livello di rete a quelli con astrazione maggiore.
IP e UDP sono due protocolli non affidabili; TCP può garantire un'affidabilità e un'ordine dei dati ma a scapito del tempo di consegna di questi.
Questo ultimo protocollo è perfetto per tutti i servizi o applicazioni che fanno uso di trasmissione dati dove è richiesta l'affidabilità e la qualità dei dati; ma per i servizi multimediali, dove il tempo di consegna dei dati stabilisce l'utilità dei dati stessi, è altamente inefficiente.
Per tentare di risolvere o minimizzare questi problemi si sono adottati vari accorgimenti negli applicativi multimediali:

- si utilizza il protocollo UDP o protocolli basati su questo per il minor consumo

di banda e di tempo per trasportare i dati

- si utilizzano dei buffer ed un ritardo iniziale per compensare il jitter
- si introducono dei timestamp per la riproduzione corretta dei dati
- si adattano le compressioni dei dati alla banda disponibile
- si utilizzando particolari modalità di invio dei dati in modo tale che le perdite di pacchetti contigui creino un deterioramento più piccolo in proporzione, o comunque un deterioramento non significativo
- si utilizzano percorsi alternativi per la trasmissione dei dati rendendo possibile la variazione del percorso dal Server al Proxy senza modificare il percorso realizzato dal Proxy al Client

1.4 Accorgimenti per la qualità del servizio

Il protocollo udp utilizza 64 bit per le informazioni del sistema, più i bit dei dati da spedire per ogni pacchetto; il protocollo tcp invece utilizza 192 bit per le informazioni del sistema più i bit dei dati. Inoltre il protocollo tcp richiede un tempo iniziale per instaurare una connessione in quanto richiede la conferma da parte del desinatario. Questi sono i motivi principali che hanno fatto si che tutti i flussi multimediali o i servizi real time si appoggiassero sul protocollo udp anziche il tcp.

Come gia accennato ci possono essere dei ritardi nell'arrivo dei pacchetti e questi ritardi possono variare; per risolvere il problema del jitter si tende ad usare un buffer e a ritardare la presentazione del flusso multimediale. Si tende ad introdurre anche dei blocchi di timestamp ovvero dei blocchi con informazioni temporali così da poter permettere al ricevente di decidere all'istante se un blocco è candidato alla presentazione o meno.
Per chiarire facciamo un piccolo esempio: se introduciamo un ritardo q nella comunicazione ed arriva un pacchetto con un timestamp del tempo t teniamo il pacchetto solamente se arriva con un tempo non superiore a t+q.
L'immagine in figura 1.2 aiuta a chiarire questo concetto.
Naturalmente il ritardo andrà accuratamente scelto in base al servizio multimediale interessato; purtroppo più grande è il ritardo, minori sono le perdite di informazioni ma minore è anche l'interattività con il servizio.

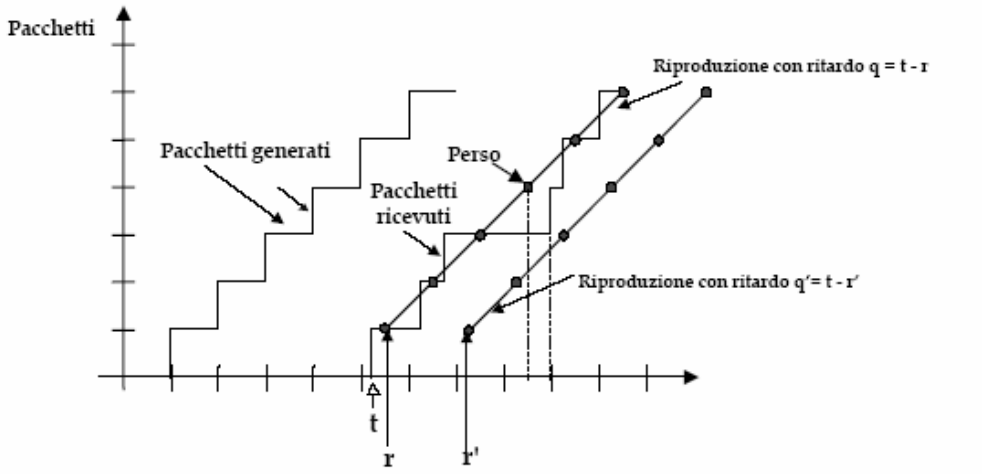


Figura 1.2: Grafici riguardanti la trasmissione tra un Server e un Client, dei pacchetti accettati e quelli scartati in relazione al tempo di arrivo.

Ci sono servizi che consentono la variazione del ritardo in modo adattativo; ad esempio una connessione telefonica dove ci sono momenti di silenzio e il traffico di dati è pressoche nullo. In questi servizi ci potranno essere ulteriori ottimizzazioni giocando su questo parametro.

Affrontiamo ora un altro problema; l'imprevedibilità e l'irregolarità delle perdite dei pacchetti.
Se i pacchetti che non arrivano a destinazione sono pochi e lontani fra di loro nel flusso l'effetto finale è quasi impercettibile; ma se avvengono delle perdite di pacchetti vicini tra loro, temporalmente, l'effetto può essere sgradevole e l'informazione può risultare compromessa.
Per prevenire le perdite dei pacchetti ci sono svariate strategie che puntano ad introdurre rindondanza e a manipolare il flusso per renderlo più robusto; illustreremo due strategie in merito.

1.4.1 Strategia a XOR

La strategià a xor consiste nell'introduzione, all'interno di un flusso, di un blocco rindondante ottenuto come XOR degli n blocchi precedenti
Il vantaggio di questa strategia è, che se viene perso un blocco degli n+1, all'istante di ricezione del blocco n+1 lo posso ricostruire.
Ci sono per contro alcuni problemi:

- La banda aumenta di un fattore 1/n (ad es. 1/10 se n= 10)

- Il ritardo fisso iniziale d deve essere tale da permettere la ricezione di almeno n+1 pacchetti (se n = 10, d > 200 msec).
- Quindi aumentando n aumenta l'efficienza ma aumenta anche il ritardo e la probabilità di avere più di una perdita in una sequenza (situazione in cui questo meccanismo diventa inefficace).

1.4.2 Strategia con doppia codifica

- Si comprimono i dati usando due codifiche a risoluzioni diverse: una bassa (B) ed una (PCM 64Kbit e GSM a 13 Kbit) alta (A) (ad es. un PCM a 64 Kbit/s e un GSM a 13 Kbit/s).
- Si costruisce il pacchetto n prendendo il blocco n del flusso codificato con A (160 Byte) e il blocco n-1 del flusso codificato con B (32,5 byte)
- Quando un pacchetto viene perso può essere ricostruito (con minore qualità) all'arrivo del pacchetto successivo
- Si può proteggere anche nei confronti di due perdite consecutive aggiungendo anche il blocco n-2 del flusso B.

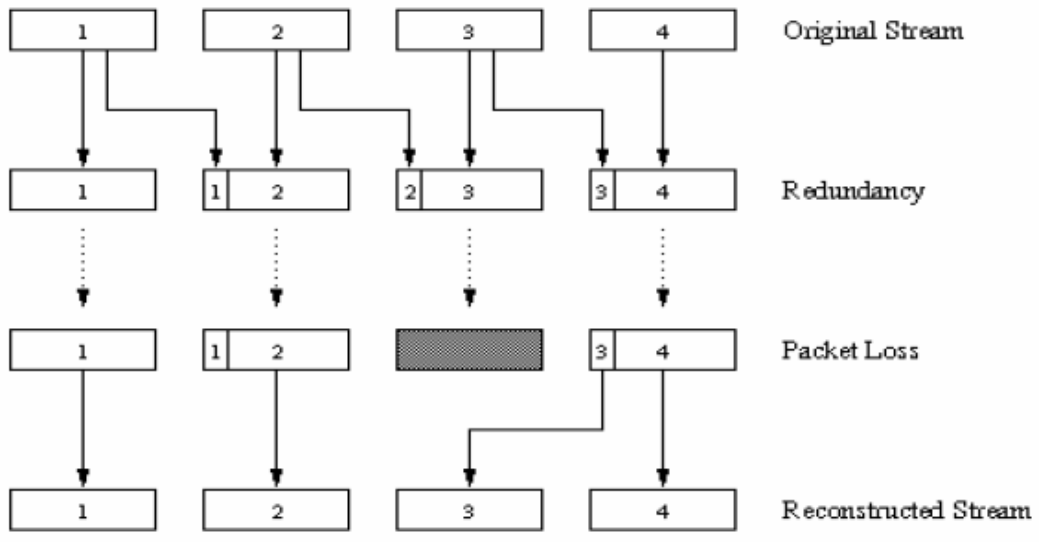


Figura 1.3: Schema di codifica con la strategia suddetta. Dall'immagine si nota la distribuzione delle informazioni relative a un pacchetto, su più pacchetti.

1.4.3 Interleaving

– I blocchi sono spezzati in unità più piccole (ad esempio in 4 unità da 40 byte).
– Le unità vengono mescolate in più pacchetti spediti in tempi successivi che quindi contengono parti di più blocchi.
– I blocchi originari vengono ricostruiti al ricevitore.
– Se un pacchetto viene perso la perdita viene distribuita su n blocchi ed il risultato è una più semplice compensazione.

1.4.4 Utilizzo dei percorsi alternativi

Un sistema con questa caratteristica è chiamato anche sistema multipath. Come mostra la figura 1.4, questo tipo di sistema, vede l'utilizzo di molteplici Server per la fornitura del servizio e un dispositivo intermedio, chiamato Proxy, decide dinamicamente quale Server utilizzare in base a criteri legati alla qualità del servizio. Il tutto senza interessare la trasmissione tra il Proxy ed il Client.

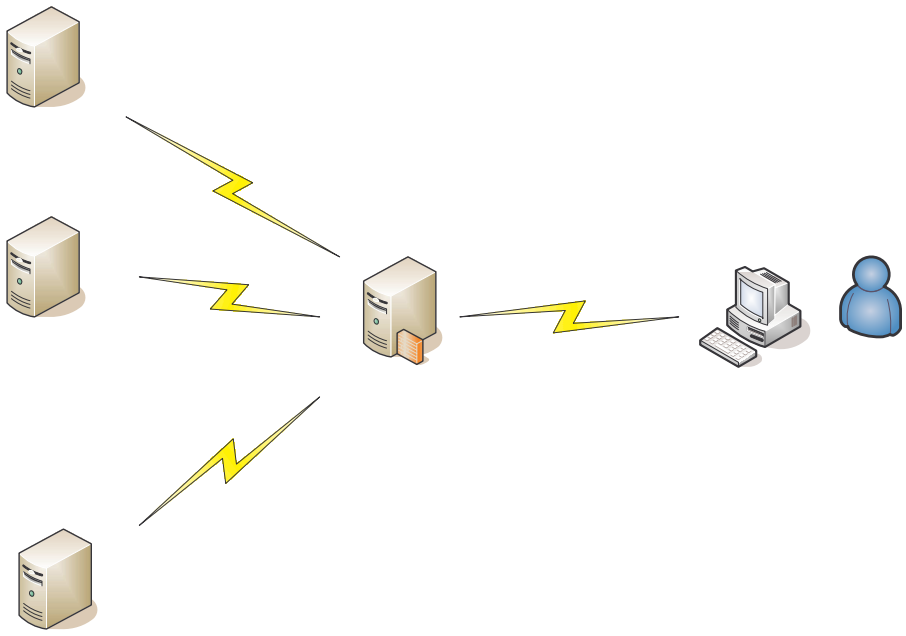


Figura 1.4: Esempio di sistema multipath dove un dispositivo intermedio tra il Client e il Server, chiamato Proxy, decide quale Server utilizzare per l'erogazione del servizio.

1.5 RTP

Con il continuo aumento di servizi multimediali su internet è andato via via crescendo il bisogno di un protocollo adatto a questo genere di servizi.
RTP è nato per questo. E' un protocollo che lavora a livello applicativo secondo lo standard OSI (Figura 1.5); solitamente utilizza udp, ma non è escluso che possa lavorare sopra il tcp ed è stato pensato per poter essere personalizzato, in base alle esigeze specifiche di ogni servizio multimediale.
Non garantisce qualità del servizio ne riserva le risorse ma fornisce informazioni utili per il controllo delle trasmissioni. Lavora affiancato dal protocollo RTCP che dettaglieremo in seguito.
La figura sottostante rappresenta il protocollo RTP in un modello OSI.

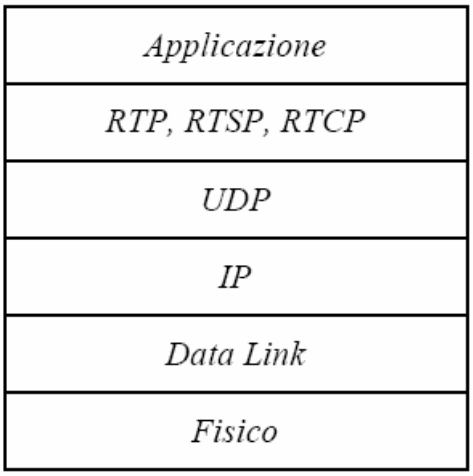


Figura 1.5: Schema protocollo RTP secondo lo standard OSI.

1.5.1 Caratteristiche del protocollo RTP

Offre le seguenti funzionalità:

- Identificazione del payload
    - Contiene l'identificativo del payload, i relativi codici vanno definiti nel profilo.
- Numeri di sequenza
    - Il valore iniziale è scelto a caso, e successivamente viene incrementato di

1 per ogni pacchetto

- Time stamp
    - Riflette l'istante di campionamento del primo ottetto del payload. Se i pacchetti sono generati periodicamente, viene considerato allora l'istante nominale di campionamento, e quindi il timestamp viene incrementato di 1 per ogni periodo di campionamento, il valore iniziale è comunque casuale. Più pacchetti possono avere lo stesso timestamp, come accade, ad esempio, se una frame video viene spedita utilizzando più pacchetti.
- Capacità di identificazione delle sorgenti per la sincronizzazione.

1.6 RTCP

Rtcp è un protocollo affiancato ad RTP e permette con le sue funzionalità di monitorare le trasmissioni e ricavare informazioni sulla qualità del servizio.

1.6.1 Caratteristiche del protocollo

- Feedback proprietà della trasmissione
    - Principalmente RTCP deve fornire un feedback sulla qualità della distribuzione dei dati, quindi permette di scambiare informazioni sul numero di pacchetti ricevuti o persi sul jitter e così via. Questo riscontro può essere utilizzato direttamente da quelle applicazioni che realizzano tecniche di codifica a tasso variabile. Queste informazioni sono trasportate nei pacchetti RTCP *sender report* e *receiver report*.
- Diffusione del canonical name
    - Ogni sorgente RTP, come abbiamo visto, è identificata da un SSRC, tuttavia questo identificativo cambia ogni volta che si fa partire l'applicazione, mentre può essere utile avere un identificativo che sia univoco e non cambi al cambiare della sessione. Per questo motivo si usa il CNAME (Canonical Name) che viene diffuso tramite RTCP.
- Controllo traffico RTCP
    - Per ogni sessione, si assume che comunque il traffico RTCP debba mantenersi al disotto di un fissato limite, per questo motivo se il numero

di partecipanti cresce ognuno dovrà ridurre il tasso di emissione di pacchetti di controllo.

- Personalizzazione trasmissione informazioni
    - Opzionalmente è possibile trasportare con RTCP alcune semplici informazioni sui partecipanti alla sessione, come ad esempio il nome, l'indirizzo e-mail, ecc.

1.7 Conclusione

Cap. 2 Java Media Framework: analisi della libreria e dei suoi limiti

2.1 Uno sguardo generale

Il JMF (Java Media Framework) è una collezione di Api (application program interface) per la gestione di contenuti multimediali all'interno di applicazioni e applet Java.

Alcune caratteristiche rendono il JMF potente e flessibile:

- è caratterizzato dalla possibilità di manipolare diversi formati di file multimediali con diverse codifiche sia audio che video;
- supporta la trascodifica dei dati in input in tempo reale da un formato a un altro e la trasmissione con vari protocolli ad una successiva destinazione;
- permette l'acquisizione da sorgente in ingresso quali videocamera o microfono, dei dati;
- ha un'architettura ad alto livello cosicchè il programmatore sia in grado di utilizzare il framework senza dover conoscere il suo funzionamento interno ed estenderlo senza troppi problemi qualora lo ritenga necessario.

Sono state create 2 versioni del JMF ed ora non è più supportato dalla Sun.

2.2 Architettura del JMF

JMF Fornisce un architettura unificata con la messaggistica di protocolli per la gestione dei processi e la trasmissione di contenuti multimediali.

Una distinzione fondamentale dei passi, che avvengono sulla manipolazione dei contenuti multimediali basati sul tempo, viene rimarcata nella figura 2.1; 3 sono le sezioni:

- La prima sezione di input dove abbiamo un file presente su disco; un file

catturato da telecamera oppure ricevuto dalla rete.

- una fase intermedia di process dove possiamo lavorare il flusso, comprimerlo, decomprimerlo convertirlo tra i formati.
- una fase finale di output dove il flusso può essere presentato, salvato su un file o spedito attraverso la rete.

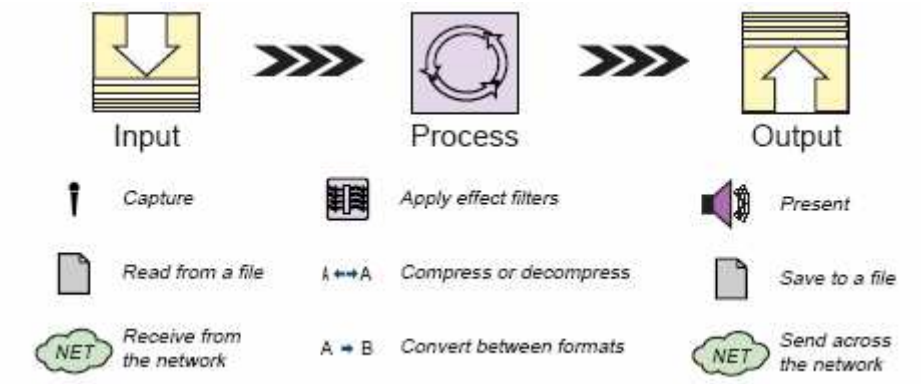


Figura 2.1: vengono mostrate le 3 fasi di lavorazione di un flusso multimediale.

Dalla presentazione di questa suddivisione in fasi nella manipolazione dei flussi multimediali passiamo a presentare il modello chiave del JMF per la gestione di queste fasi.

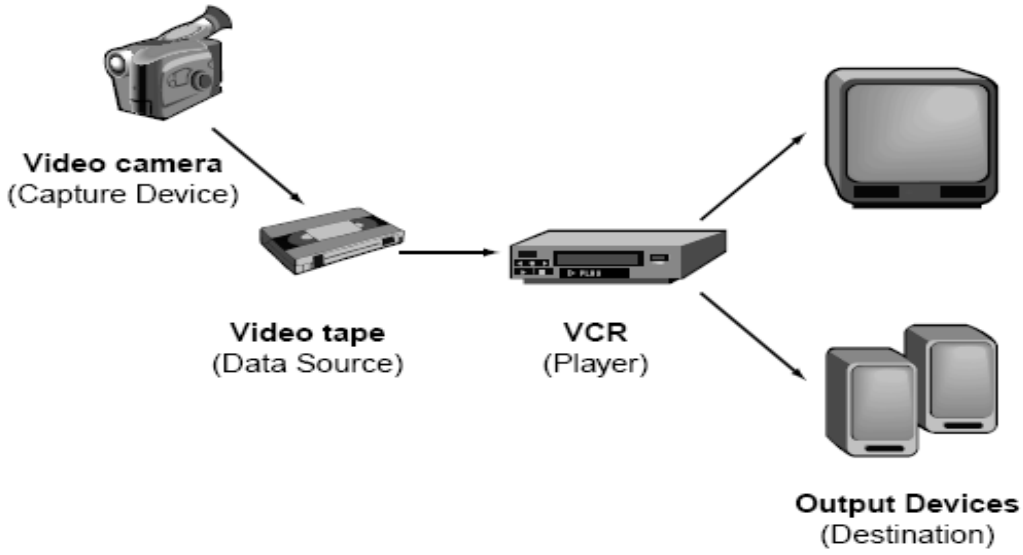


Figura 2.2: Modello di base del JMF per la manipolazione di un flusso multimediale

Dal modello rappresentato in figura 2.2 possiamo comprendere come il JMF tratti i vari dispositivi:

- Ogni periferica di acquisizione sia audio che video è vista come un oggetto "*CaptureDevice*" che fornisce stream audio o video elementari da processare
- L'astrazione di ogni sorgente dati viene vista come un oggetto chiamato *DataSource*.
- Il visualizzatore dei dati multimediali, qualsiasi formato essi abbiano, è visto come un oggetto chiamato *Player***.** Questo dispositivo processa il DataSource, estrae da eventuali flussi multiplexati le singole tracce ; da ogni traccia decodifica il flusso (audio o video) e trasmette i flussi cosiddetti elementari al video e alle casse acustiche.

Il modello soprastante è particolare al caso si voglia presentare il contenuto multimediale; in sostituzione al Player, per le situazioni dove ciò non è desiderato, esiste un oggetto chiamato *Processor* che consenste di poter manipolare i flussi, applicare filtri, cambiare formato e fornire in uscita un nuovo DataSource da poter utilizzare in maniera diversa dalla presentazione.
La figura sottostante ci permette di capire ciò.

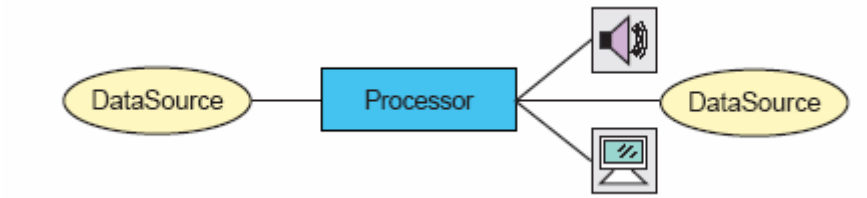


Figura 2.3: Mostra le Api per il processing dei dati con il JMF. Viene utilizzato il Processor che da una sorgente di dati ne crea un'altra.

Le Api del JMF ruotano intorno a quattro gestori:

- **Manager**

  Gestisce la costruzione di Player, Processor, DataSource e DataSink. Questo livello di indirezione permette a nuove implementazioni di essere integrate senza problemi con il JMF

  Da una prospettiva del cliente questi oggetti sono sempre creati nello stesso modo; l'oggetto viene costruito da un implementazione standard o da una personalizzata.

- **PackageManager**

  Mantiene un registro dei package che contengono classi JMF come Player Processor DataSource o DataSink

- **CaptureDeviceManager**

  Mantiene un registro delle periferiche di cattura disponibili.

- **PlugInManager**

  Mantiene un registro di tutti i PlugIn del JMF come Multiplexer Demultiplexer Codecs Effects Renders.

2.2.1 Modello ad eventi del JMF

JMF usa un meccanismo a segnalazione di eventi per tener informati i programmi, basati sul questo framework, interessati allo stato dei flussi multimediali; consente loro di rispondere e gestire condizioni di errore come mancanze di dati o risorse non disponibili.

Quando un'Api del JMF ha bisogno di riportare il proprio stato spedisce un MediaEvent.

MediaEvent è la radice di tutti gli eventi riguardanti appunto i Media che lavorano con questo framework; ogni oggetto JMF che può spedire MediaEvent definisce un corrispondente ascoltatore di eventi (Figura 2.4).

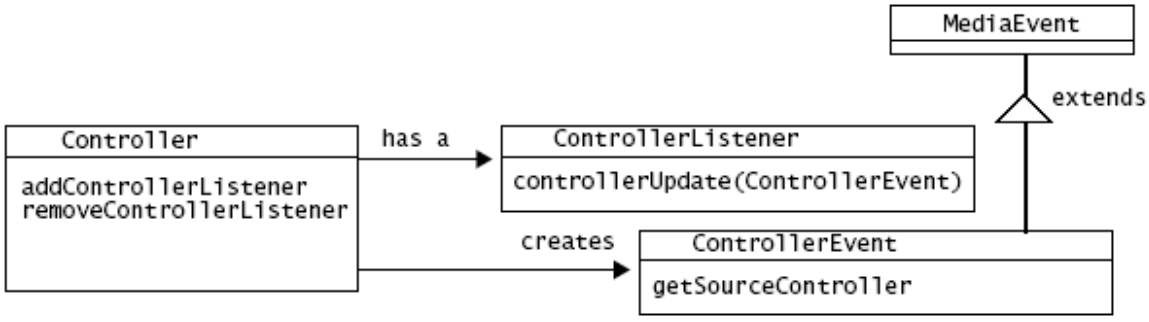
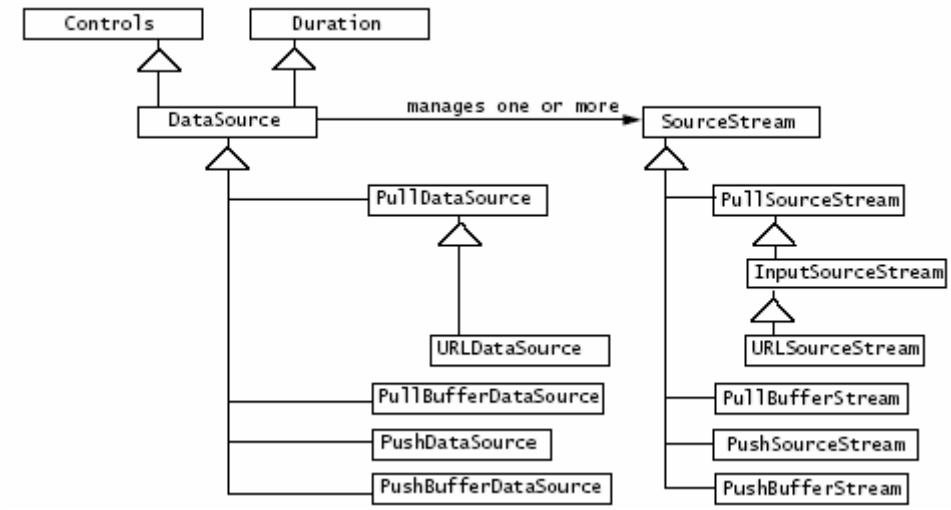


Figura 2.4: Schema del modello ad eventi del JMF. L'oggetto che spedisce eventi è un controller. Gli oggetti che vogliono ricevere l 'evento sono devono estendere ControllerListener e registrarsi presso il Controller.

2.2.2 Modello dei dati

I Player JMF usano DataSources per gestire il trasferimento dei contenuti multimediali.
Un DataSource è un Api per l'astrazione di tutti i sorgenti dati: maschera il protocollo e il software usato per la trasmissione del flusso.
Un DataSource viene identificato da un MediaLocator JMF (descrittore per il contenuto di un media) o da un URL da cui si può comunque costruire un MediaLocator; gestisce internamente un elenco di oggetti SourceStream (Figura 2.5) che possono usare un elenco di Byte come unita di trasferimento o un Buffer appropriato a seconda del tipo di DataSource.

Figura 2.5 : Schema dei tipi di DataSource esistenti nel JMF.



I flussi o stream possono essere distinti a seconda delle caratteristiche di trasmissione:

- *pull*, se la trasmissione è avviata e gestita dal Client. Sono protocolli di questo tipo, ad esempio, il protocollo FILE ed il protocollo HTTP;
- *push*, se invece è il Server ad iniziare e controllare la trasmissione. E' di questo tipo, ad esempio, il protocollo RTP.

Entrambi i tipi di stream possono essere bufferizzati per ottimizzare il flusso di trasmissione.

2.2.3 Formato dei dati

Per rappresentare il formato dei dati, JMF usa un oggetto Format che non contiene dettagli su tempi o codifiche ma solo informazioni sul tipo di formato quali il nome della codifica del formato e il tipo di dati che il formato richiede.

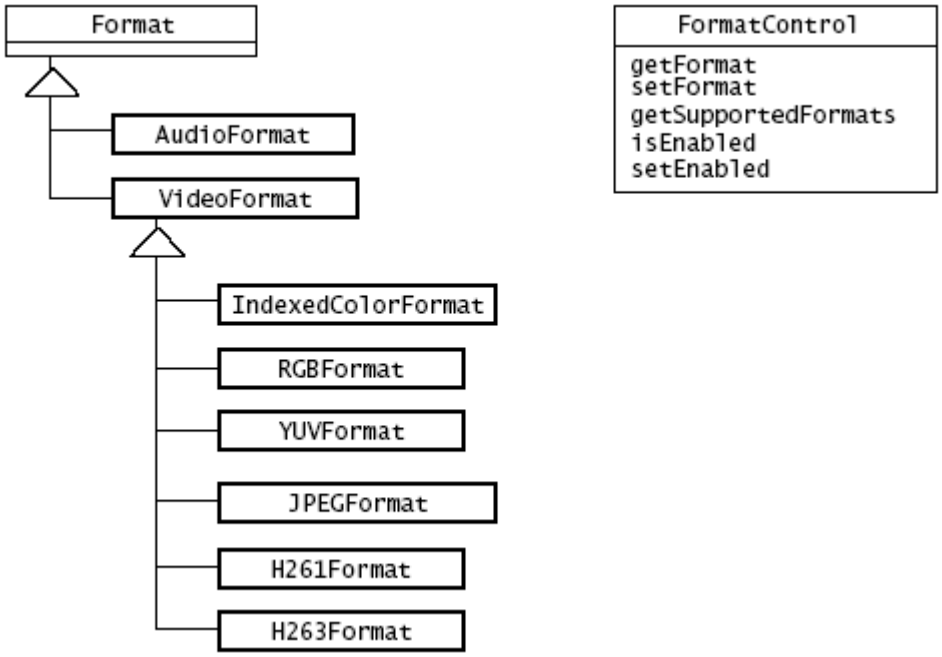


Figura 2.6 : Schema formato dei dati del JMF. Tramite un'interfaccia FormatControl si può capire quale formato dei dati si sta utilizzando.

2.2.4 Controlli del JMF

JMF Control fornisce un meccanismo per l'impostazione e l'interrogazione sugli attributi di un controllo.
Un controllo fornisce spesso accesso al componente di tipo interfaccia utente che permette di controllare gli attributi dell'oggetto.

JMF definisce un'interfaccia standard di Control che ha come unico metodo
java.awt.Component getControlComponent();
Il JMF contiene, di standard, le interfacce per i controlli mostrati in figura:

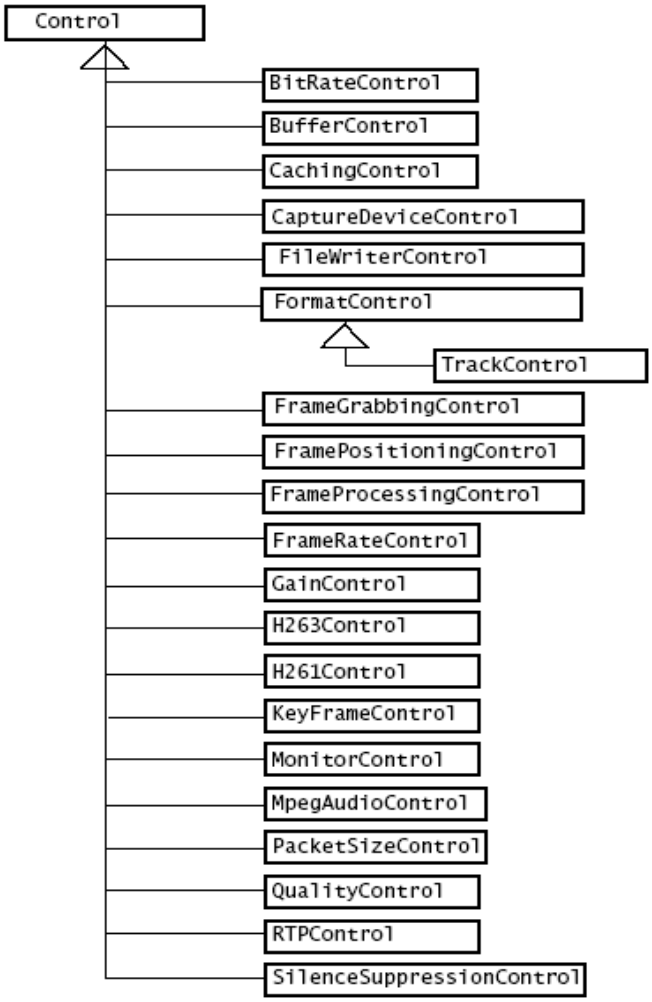


Figura 2.7 : I controlli standard del JMF. Tramite un'interfaccia si può ottenere da ogni oggetto i suoi controlli e si possono visualizzare.

I controlli più comuni sono:

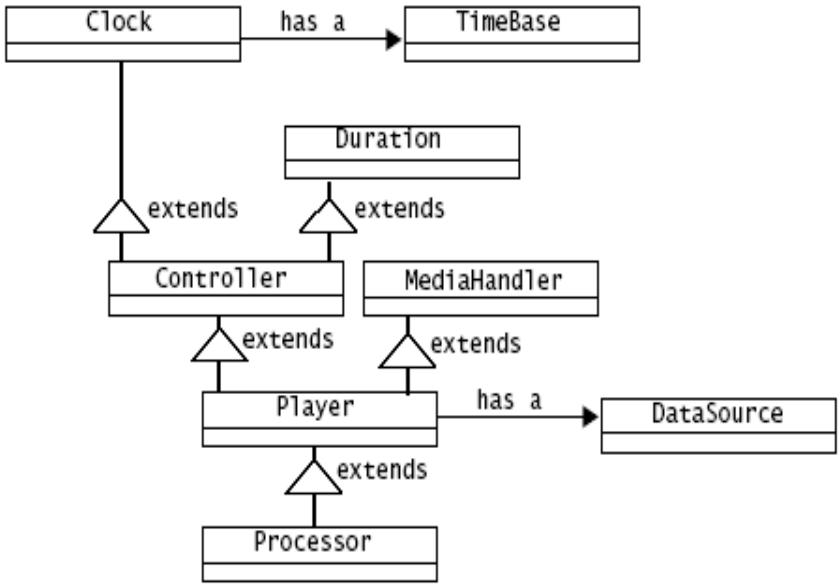
- CachingControl Consente il monitoraggio del download dei file.
- GainControl permette di controllare il volume dell'audio all'uscita del Player o del Processor.

Oggetti DataSink che leggono un flusso da un DataSource e lo portano su file implementano il controllo FileWriterControlInterface. Questo controllo permette all'utente di limitare la dimensione del file.

2.3 Presentazione dei flussi multimediali.

In JMF il processo di presentazione è modellato dall'interfaccia Controller. Lo schema ad ereditarietà mette in risalto le interfacce importanti.

Figura 2.8: I Controller del JMF.



Le Api JMF definiscono 2 tipi di controller : *Player* e *Processor*.
Sia il Player che il Processor normalmente sono costruiti per specifici DataSource e non possono essere riutilizzati per presentare altri media.
Questo perché, come vedremo in seguito, ad ogni DataSource corrisponde uno specifico MediaHandler chè è contenuto nel Player.
Passiamo al dettaglio dei due controller.

2.3.1 Player

Il Player è un'Api per processare i flussi multimediale in ingresso e renderizzarli a precisi istanti; la destinazione della renderizzazione dipende dal tipo di media che si è andati a processare.
Un Player non fornisce controlli relativi alla processione dei dati o alla renderizzazione di essi.
E' costituito da vari stati e si trova sempre in uno e uno solo di essi. Nella figura sottostante sono rappresentati gli stati e i possibili spostamenti da uno stato all'altro.

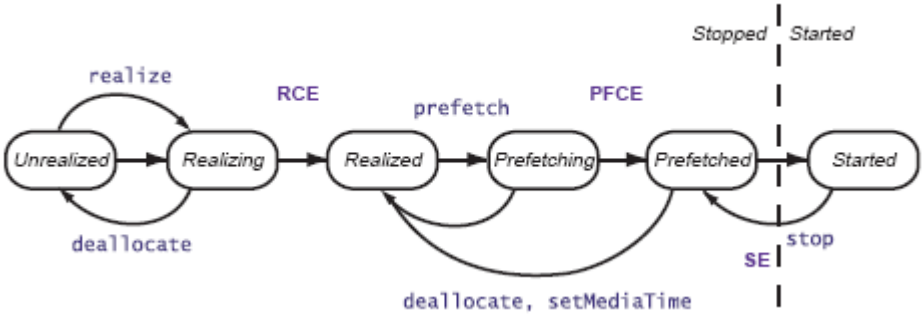


Figura 2.9 : Stati del Player.

Gli stati a sinistra della linea tratteggiata costituiscono le varie fasi di creazione del Player.
L'ultimo stato chiamato "*prefetched*" è lo stato dove il Player è completamente costruito e varia da questo stato allo stato started a seconda del funzionamento o meno; se viene fatto partire va allo stato "*started*" e se viene fermato si riporta allo stato "*prefetched*".

Esempio di creazione di un Player: Tramite il Manager si ottiene il Player adeguato al tipo di media e si realizza il player chiamando il metodo realize; terminata la realizzazione del Player si possono ottenere i controlli visuali che ne permettono la gestione.

```
Player p = Manager.createPlayer(new MediaLocator("file://.."));
p.realize();
while (p.getState()!=p.Realized);
Component comp = p.getVisualComponent();
p.start();
```

2.3.2 Processor

Un Processor è una specializzazione della classe Player; può pertanto fare tutto quello che un Player consente ma, in aggiunta, fornisce un flusso multimediale attraverso un DataSource che può essere presentato da un altro Player, Processor o spedito attraverso un DataSink.
Mentre lo svolgimento del Player è predefinito, il Processor consente allo sviluppatore dell'applicazione di personalizzare il tipo di trattamento da applicare ai dati multimediali in ingresso.

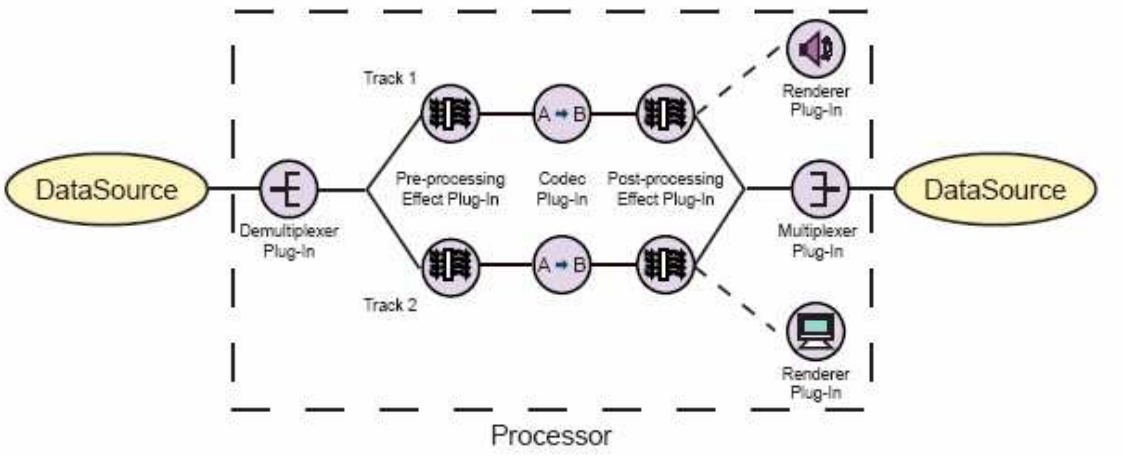


Figura 2.10 : Schema dettagliato delle possibilità del processor di manipolazione dei flussi. Il processor è in grado di cambiare la codifica delle singole tracce elementari o il formato di pacchettizzazione del flusso.

Per quanto riguarda gli stati si differenzia dal Player per l'aggiunta di 2 stati segnati in rosso nella figura 2.11.

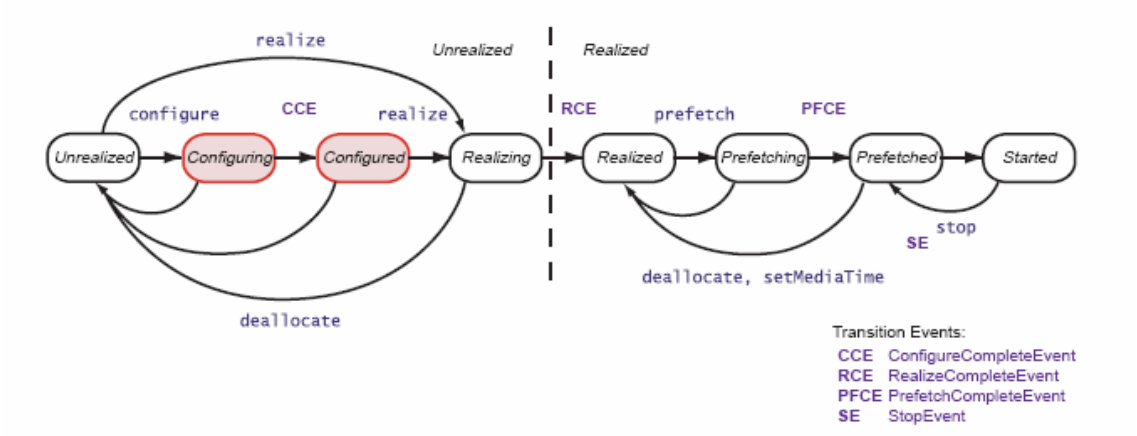


Figura 2.11 : Stati del processor. A differenza del Player, il processor deve essere configurato.

2.3.3 DataSink

DataSink è l'interfaccia di base per gli oggetti che leggono i flussi multimediali dal DataSource e lo reindirizzano ad una destinazione diversa dal Player.
Consentono quindi il salvataggio dei media sui file o la trasmissione attraverso la rete.

2.4 Trasmissione dei flussi multimediali attraverso la rete

Il JMF come già accennato è predisposto per trasmettere i flussi multimediali attraverso la rete con diversi protocolli.
DataSink è l'interfaccia utilizzata per fare ciò ma servono altre Api per poter lavorare con i vari protocolli.
Nei prossimi paragrafi tratteremo il protocollo RTP perchè è sull'ottimizzazione di questo che si basa la tesi.

2.4.1 RTP

Il JMF permette la manipolazione dei contenuti multimediali con il protocollo RTP attraverso apposite Api presenti nel package javax.media.rtp.
Le figure sottoindicate (Figura 2.12 e 2.13) mostrano rispettivamente lo schema degli oggetti nella ricezione di un flusso tramite protocollo RTP e la trasmissione di un flusso con il medesimo protocollo.

Figura 2.12 :Schema per la ricezione

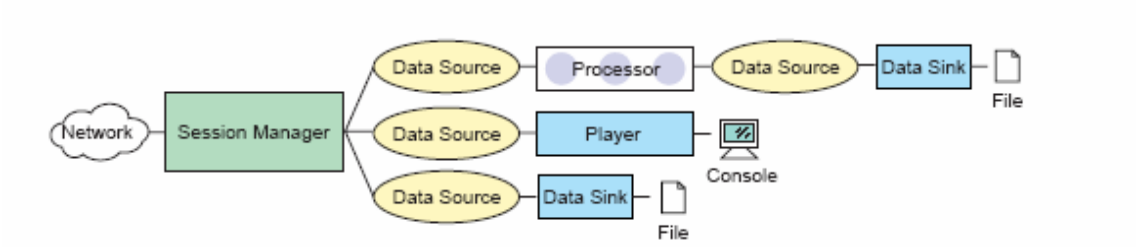
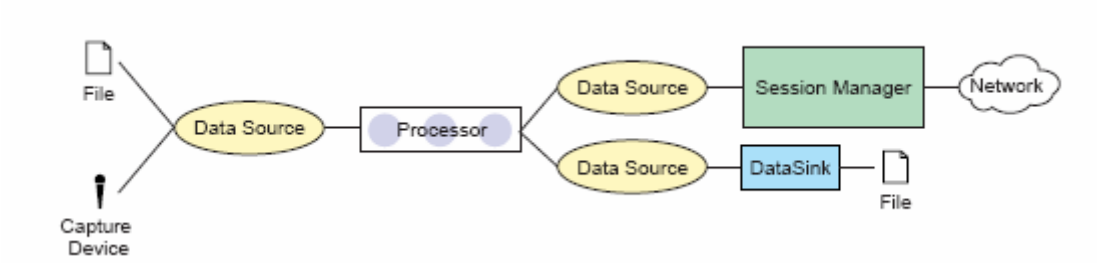


Figura 2.13 :Schema per la trasmissione

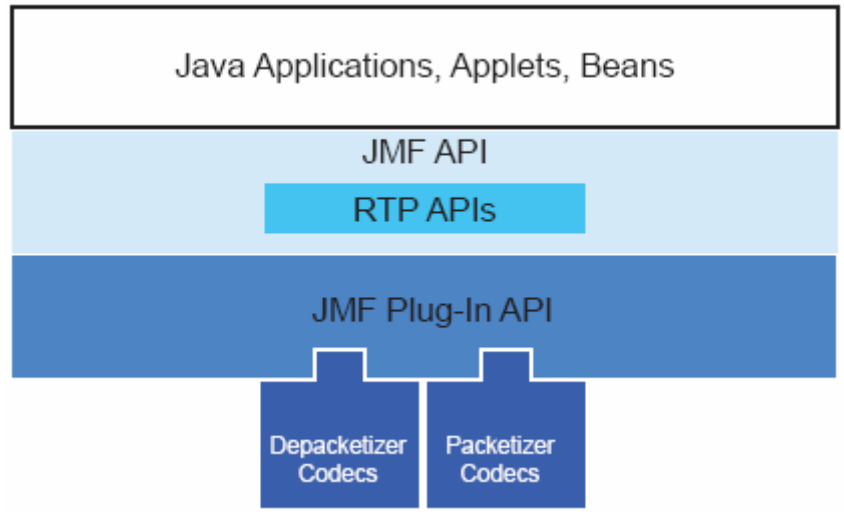


Dagli schemi si nota un oggetto chiamato SessionManager che si interpone tra la rete e il DataSource. Approfondiremo questo oggetto nel paragrafo successivo.

2.4.1.1 Architettura RTP nel JMF

Le Api RTP del JMF sono progettate per interagire con dispositivi di input, di presentazione e di manipolazione dei flussi. Si possono trasmettere dati ricevuti da dispositivi di input o immagazzinare i dati ricevuti su file.
JMF può estendere i formati RTP e payload attraverso il meccanismo offerto a PlugIn.

Figura 2.14 :Architettura ad alto livello RTP del JMF



Ogni trasmissione RTP nel JMF è contraddistinta da una sessione.
Una sessione RTP è identificata da un indirizzo internet e una coppia di porte; una utilizzata per i dati e una utilizzata per il controllo.
Per ottenere una sessione si utilizza un apposito gestore chiamato SessionManager.

2.4.1.2 Session Manager

Nel JMF il SessionManager è utilizzato per coordinare le sessioni RTP; tiene traccia dei partecipanti alle sessioni, degli stream trasmessi e di tutte le statistiche correlate.
Una volta registrato un DataSource per l'attesa di un trasmissione con il SessionManager si attende l'evento di ricezione dei dati e quindi la creazione della Sessione.

2.4.1.3 Eventi Rtp

La gestione di flussi trasportati con protocollo RTP avviene attraverso il meccanismo degli eventi.
La figura sottostante riporta tutti gli eventi correlati con le Api RTP.

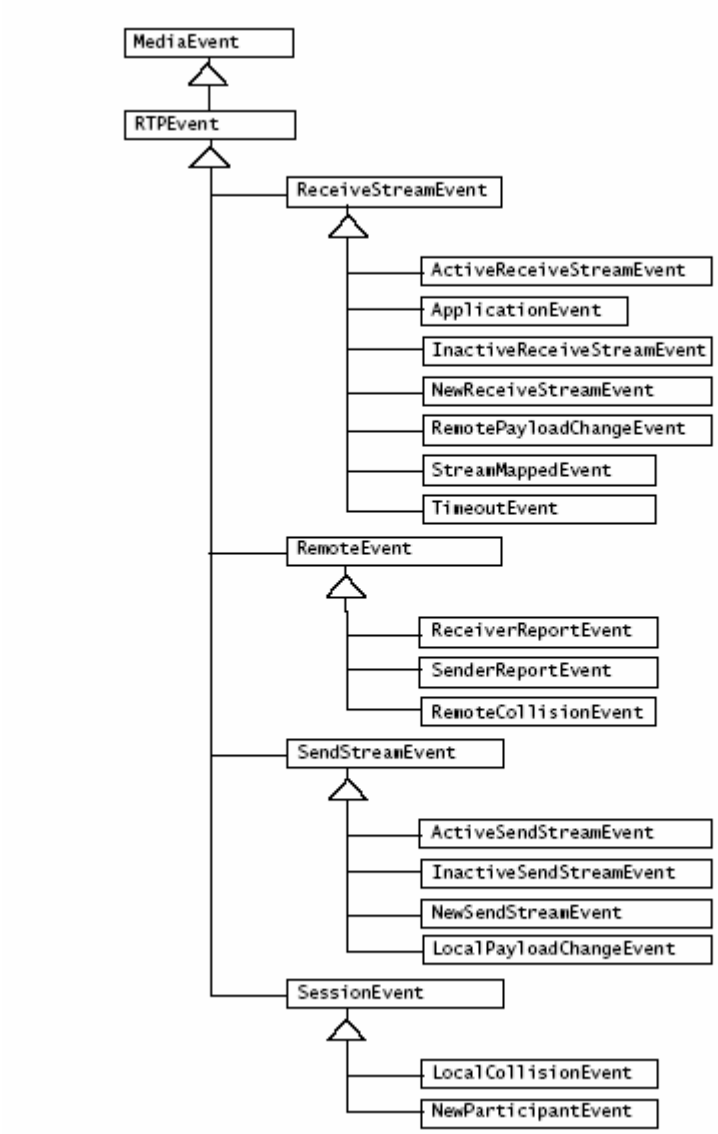


Figura 2.15 : Gli eventi scatenati dalle Api rtp. Ogni gruppo di eventi interessa una situazione particolare con l'utilizzo del protocollo RTP.

Gli eventi di maggior interesse nel nostro caso sono quelli riguardanti la ricezione; ovvero gli eventi che specializzano *ReceiveStreamEvent*.
Nella realizzazione di un Proxy con cambio dinamico del DataSource sarà necessaria la gestione efficiente di questi eventi.

2.5 Limiti del JMF

Nonostante il JMF abbia un ottimo meccanismo a plug-in per l'estensione soffre comunque di rigidità strutturali :

- ogni sessione di manipolazione di un flusso multimediale è indipendente dalle altre:

  ogni volta che viene presentato, processato un flusso multimediale, vengono allocati tutti gli oggetti necessari e non è possibile riutilizzare gli oggetti già allocati per la presentazione, processazione di un altro flusso multimediale.

  Questo porta in molte situazioni inevitabilmente a degli sprechi di risorse.
- non è possibile ottenere la personalizzazione di Api di un livello di astrazione se queste richiedono modifiche alle Api di altri livelli:

  il meccanismo a plug-in è ottimo nella personalizzazione dei codec o packetizer dove i metodi resteranno sempre gli stessi e cambierà solo l'implementazione; è d'altro canto scomodo nel Player, Processor se si vogliono aggiungere funzionalità che richiedano l'accesso alle Api di livello inferiore.
- non è possibile personalizzare il motore di gestione del Player e del Processor in quanto tutte le Api sono costruite per lavorare con un unico motore.

  Il problema accennato nel verso precedente è collegato con questo problema, ovvero la non previsione di una possibile personalizzazione del motore di gestione del Player o del Processor; il Manager adibito alla creazione automatica del Player o del Processor è costruito per lavorare con un solo motore ovvero PlayBackEngine per il Player e ProcessEngine per il Processor.

  Questo è il problema più importante perchè mette in risalto una scelta decisa e costosa, in termini strutturali, dei progettisti del JMF.

Per la realizzazione di un'architettura per l'erogazione dei servizi, con caratteristica a multipath, il JMF così com'è non può offrire una grande qualità del servizio perchè richiederebbe ad ogni cambio di Server da parte del Proxy un consumo di risorse inutile dovuto al fatto che ogni sessione di trasmissione è indipendente e si devono riallocare tutti gli oggeti interessati: uno spreco di memoria assurdo.

Le modifiche per permettere un utilizzo intelligente delle risorse potrebbero essere fatte o sul gestore del flusso alla radice (DataSource) oppure sul Demultiplexer che gestisce il DataSource .
A causa della mancanza del codice sorgente relativo alle Api di ricezione dei flussi con protocollo RTP, lo studio si è concentrato sulla modifica del Demultiplexer che per le rigidità del JMF sopra dettagliate, ha richieto le modifiche alle Api dei livelli soprastanti sino al Player ed al Processor.

2.6 Approfondimenti del JMF per il progetto

Ora approfondiremo alcuni oggetti del JMF per poter comprendere appieno lo studio effettuato e il progetto svolto.

2.6.1 DataSource

Il DataSource, come già detto, è l'oggetto che astrae il sorgente del flusso multimediale.
Abbiamo parlato dell'esistenza di quattro tipi di DataSource.
Il tipo *PushBufferDataSource* è il DataSource che caratterizza i dati rapprentanti i flussi trasferiti attraverso la rete con protocollo RTP e necessita di un breve approfondimento in quanto è oggetto di studio per il progetto.
PushBufferDataSource contiene al suo interno oggetti di tipo *PushBufferStream* che rappresentano i singoli flussi.
Questo tipo di flusso multimediale, nel JMF, viene gestito tramite un thread specifico che notifica a un gestore dei dati quando questi possono essere trasferiti nel Demultiplexer.
E' il DataSource (da thread indipendente) a gestire il trasferimento dei dati tramite un oggetto che guida questo trasferimento e notifica al Demultiplexer quando i dati sono disponibili.
L'oggetto interessato implementa l'interfaccia BufferTransferHandler che ha un unico metodo per effettuare il trasferimento dei dati.

```
BufferTransferHandler void transferData(PushBufferStream stream)
```

2.6.2 Player

Il Player, oggetto responsabile della manipolazione dei dati dall'input (tramite DataSource) all'output, viene creato tramite il gestore *manager* che, a seconda del tipo di protocollo, crea il Player adeguato.

2.6.2.1 Dettagli sulla creazione del Player

La creazione del Player avviene attraverso un meccanismo di ricerca di un oggetto di tipo MediaHandler adeguato (già accennato nei paragrafi precedenti). Questo oggetto viene cercato da un elenco fornito dal packageManager.
Viene fornita una lista di nomi di oggetti; l'algoritmo di creazione del Player tenta di istanziare ad uno ad uno le classi e di impostare il DataSource tramite un metodo apposito.
Se l'esecuzione del metodo va a buon fine senza eccezioni l'algoritmo termina restituendo il Player.
Questa tecnica viene adottata anche per l'istanziazione di oggetti più interni di cui dopo parleremo.
Il codice sottostante relativo alla funzione manager.CreatePlayer(...) aiuta a comprendere il meccanismo sopra citato.

```
Enumeration protoList = getDataSourceList(
  sourceLocator.getProtocol()).elements();

while(protoList.hasMoreElements()) {

  String protoClassName = (String)protoList.nextElement();
  DataSource source = null; //
  try {
  if ((source = (DataSource)sources.get(protoClassName))== null )
  {
    Class protoClass = getClassForName(protoClassName);
    source = (DataSource)protoClass.newInstance();
    source.setLocator(sourceLocator);
    source.connect();
  } else
    sources.remove(protoClassName);
```

Come si può notare dal codice se avviene un'eccezione dovuta all'inesistenza della

classe o per l'impossibilità di impostare il sorgente si passa ad esaminare la classe successiva altrimenti si esce dal ciclo restituendo il Player creato al chiamante.
L'istruzione interessante è l'impostazione del DataSource attraverso l'istruzione `source.setLocator(sourceLocator);` in questa chiamata viene impostato il Demultiplexer adeguato.
E' questo che rende ogni Player unico per un tipo di media. In relazione al tipo di media corrisponde un tipo di Demultiplexer. Questa impostazione non avviene direttamente dentro il Player ma tramite un oggetto molto importante; PlayBackEngine.
Questo oggetto è contenuto nel Player e rappresenta il motore di gestione vero e proprio. E' questo oggetto che si occupa di svolgere l'aggancio del Demultiplexer e delle fasi successive per la manipolazione del media.

2.6.3 Processor

La creazione del Processor avviene in maniera analoga al Player.

2.6.3.1 Dettagli sulla creazione del Processor

Il manager, ottenuto un elenco di Processor disponibili, tenta di istanziarli e di impostare il DataSource.
Anche qui possiamo osservare il codice per una migliore comprensione.

```
Enumeration protoList =
getDataSourceList(sourceLocator.getProtocol()).elements();
while(protoList.hasMoreElements()) {
  String protoClassName = (String)protoList.nextElement();
  DataSource source = null;
  try {
    if ((source = (DataSource)sources.get(protoClassName)) == null) {
      Class protoClass = getClassForName(protoClassName);
      source = (DataSource)protoClass.newInstance();
      source.setLocator(sourceLocator);
      source.connect();
    } else
    sources.remove(protoClassName);
```

Anche per il Processor esiste un oggetto analogo al PlayBackEngine chiamato ProcessEngine che si occupa di impostare il DataSource.

Cap. 3 Modifiche del JMF

Questo progetto, per mancanza di documentazione delle Api a basso livello del JMF, è stato suddivisto, in termini di risorse, per il 60% allo studio del codice del JMF e per il rimanente 40% alla progettazione e modifica di alcune parti del framework e alla documentazione.
Le modifiche apportate al JMF sono servite per aggiungere funzionalità di cambio dinamico del DataSource al Player e al Processor per media trasmessi con protocollo RTP.

3.1 Il package com.sun.media

Le classi del package *com.sun.media* sono state replicate in un altro package (com.advanced.media) e modificate per poter lavorare con un engine (PlayBackEngine e ProcessEngine) diverso.
Questo perchè, come già discusso nel capitolo precedente, tutte le Api del JMF sono legate all'oggetto PlayBackEngine (ProcessEngine per il Processor) appartenente al package com.sun.media e non consentono un engine diverso.

3.2 Livelli di modifica

Nella struttura delle Api del JMF abbiamo apportato modifiche dal basso livello sino all'alto:

- per il basso livello sono state modificate le classi RawBufferParser e BasicSourceModule
- per il livello medio sono state modificate le classi PlayBackEngine e ProcessEngine
- per il livello alto sono state modificate le classi MediaPlayer e MediaProcessor

3.3 Approfondimento sulle modifiche

Per chiarire ulterioriormente la struttura delle classi abbiamo rappresentato due schemi UML con i vincoli di ereditarietà/specializzazione.
Nella figura 1 sono rappresentate le classi di alto e medio livello; nella figura 2 sono rappresentate le classi di basso livello.

Figura 3.1 : Schema UML delle classi di alto e medio livello

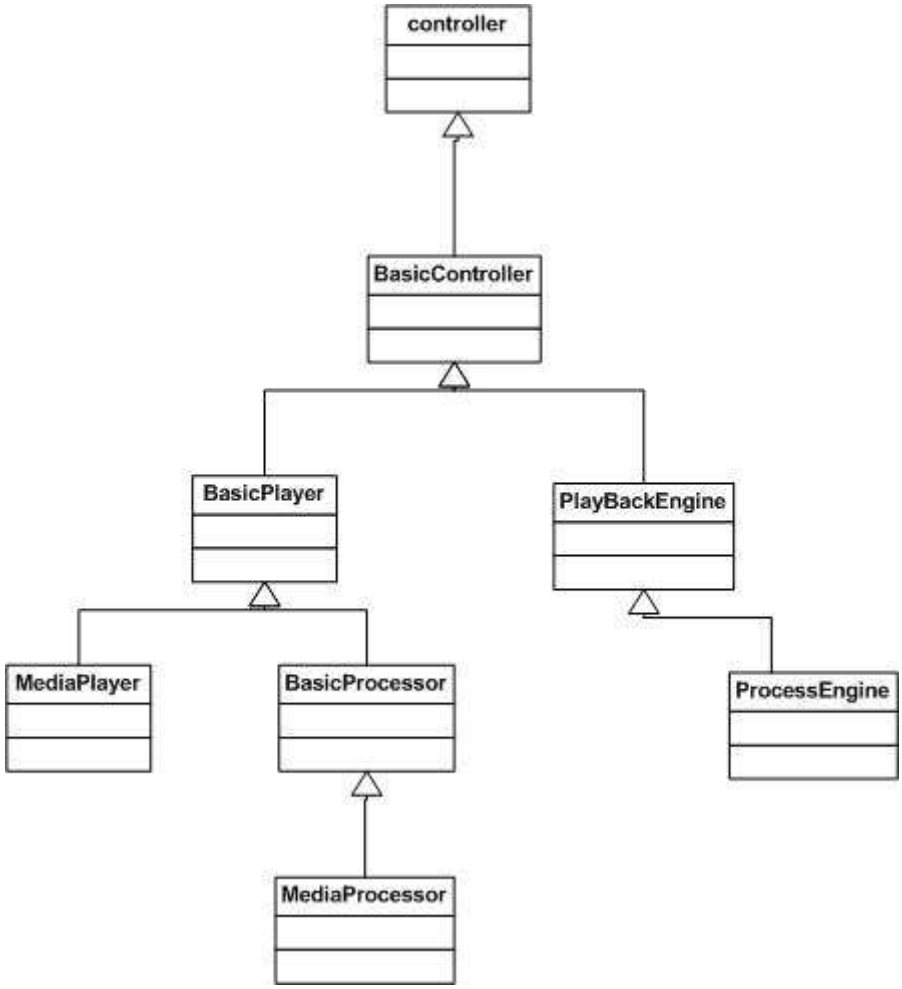
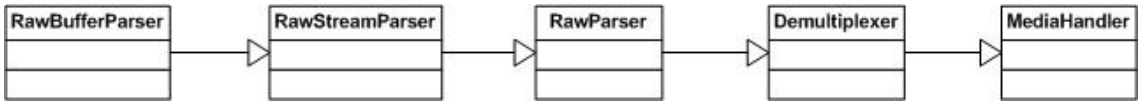


Figura 3.2 :Schema UML per chiarire la posizione del RawBufferParser e del Demultiplexer



Le classi importanti sono PlayBackEngine e ProcessEngine perchè si occupano della selezione del Demultiplexer in relazione al flusso di streaming da manipolare.
Questa selezione avviene grazie all'uso dell'oggetto BasicSourceModule attraverso il metodo createDemultiplexer(DataSource ds) che, con il medesimo algoritmo di creazione del Player e del Processor spiegati nel capitolo precedente, cerca un Demultiplexer tra quelli disponibili e non appena ne trova uno compatibile lo restituisce all'oggetto chiamante.
Analizziamo nel dettaglio le classi modificate.

3.3.1 MediaPlayer

MediaPlayer rappresenta l'Api a livello più alto per la manipolazione del Player.
In questa classe è stato aggiunto il metodo changeSource(..) e due funzioni per agganciare gli eventi ChangeSourceCompleteEvent e ChangeSourceFailedEvent.
I metodi richiamano i rispettivi metodi della classe PlayBackEngine. Riportiamo il codice aggiunto.

```
public void changeSource(DataSource source) throws
IOException,ChangeSourceException{
  engine.changeSource(source);
}

public void addChangeSourceListener(ChangeSourceListener listener){
  engine.addChangeSourceListener(listener);
}

public void remChangeSourceListener(ChangeSourceListener listener){
  engine.remChangeSourceListener(listener);
}
```

3.3.2 MediaProcessor

MediaProcessor è la rispettiva classe di gestione del Processor.
Anchessa contiene l'aggiunta del metodo changeSource(..) e l'aggancio degli eventi interessati nel cambio del sorgente.
I metodi richiamano i rispettivi metodi della classe ProcessEngine.

```
public void changeSource(DataSource source) throws
IOException,ChangeSourceException{
  engine.changeSource(source);
    }

public void addChangeSourceListener(ChangeSourceListener listener){
  engine.addChangeSourceListener(listener);
         }

public void remChangeSourceListener(ChangeSourceListener
listener){
  engine.remChangeSourceListener(listener);
         }
```

3.3.3 PlayBackEngine

La classe *PlaybackEngine* è stata modificata in alcune funzioni per la sostituzione del DataSource e la registrazione della notifica all'evento di conferma dell'operazione di sostituzione .

Riportiamo il codice significativo

```
public void changeSource(DataSource source) throws IOException{
  try{
    if (parser instanceof IDataSourceChangeable){
      ((IdataSourceChangeable)parser).changeSource(source);
      return; }
  }
  catch(IncompatibleSourceException ex){}
}


public void SendCompleteChangeSourceEvent(){
  Debug("SendCompleteChangeSourceEvent");
  for(int i=0; i< changeSourceListeners.size();i++)
    ((ChangeSourceListener)changeSourceListeners.get(i)).
      ChangeSourceUpdate(new ChangeSourceCompleteEvent(this));
}


public void SendFailedChangeSourceEvent(){
  Debug("SendFailedChangeSourceEvent");
  for(int i=0; i< changeSourceListeners.size();i++)
  ((ChangeSourceListener)changeSourceListeners.get(i)).
    ChangeSourceUpdate(new ChangeSourceFailedEvent(this));
}

public void addChangeSourceListener(ChangeSourceListener listener){
  changeSourceListeners.add(listener);
}

public void remChangeSourceListener(ChangeSourceListener listener){
  changeSourceListeners.remove(listener);
}
```

3.3.4 ProcessEngine

La classe ProcessEngine non è stata modificata perchè estendendo la classe PlayBackEngine ha ereditato le modifiche.

3.3.5 BasicSourceModule

La classe *BasicSourceModule* si occupa di selezionare il Demultiplexer corretto in base al contenuto del DataSource.
E' stata creata una seconda funzione per la selezione del Demultiplexer che in presenza di DataSource di tipo PushBufferDataSource (quali DataSourceRTP) seleziona un Demultiplexer alternativo.
Questa soluzione ovviamente non è unica; si può ottenere l'aggancio del multiplexer desiderato registrandolo nel Registry del JMF come Plugin Demultiplexer.

```
static public BasicSourceModule createModule(DataSource ds)
  throws IOException, IncompatibleSourceException {
  //Demultiplexer parser = createDemultiplexer(ds);
  Demultiplexer parser = createDemultiplexer2(ds);
  if (parser == null)
    return null;
  return new BasicSourceModule(ds, parser);
}

static protected Demultiplexer createDemultiplexer2(DataSource ds)
  throws IOException, IncompatibleSourceException {
  Demultiplexer parser = null;
  if (ds instanceof PushBufferDataSource)
    parser = new RawBufferParser();
  if (ds instanceof PushBufferDataSource){}
  if (ds instanceof PushBufferDataSource){}
  if (ds instanceof PushBufferDataSource){}
  try {
    parser.setSource(ds);
    }
  catch (IOException e) {
    parser = null; }
```

```
  if (parser == null)
    parser = createDemultiplexer(ds);
    return parser;
 }
```

3.3.6 RawBufferParser

La classe RawBufferParser che rappresenta un Demultiplexer per i DataSource di tipo PushBufferDataSource è stato modificata affinche potesse sostituire gli oggetti *streams* che costituiscono i canali del DataSource dove si prelevano i dati.
Si è dovuto modificare il Demultiplexer in quanto è all'interno di questo oggetto che avviene la registrazione del gestore del DataSource e non è possibile controllarla altrove.
Questa caratteristica è tipica dei DataSource di tipo PushBufferDataSource dove il controllo del flusso di dati avviene nella classe che riceve i dati RTPSourceStream che nel nostro caso, non avendo i sorgenti disponibili, non potevamo modificare.
All'interno della funzione changeSource effettuiamo controlli sul tipo di DataSource e sul numero di stream che possiede il nuovo DataSource.
Tuttuavia la sostituzione di DataSource contenenti formati di flussi con codifica RTP diversa può dar luogo a problemi nel decodificatore.
Grazie alle possibilità offerte dal manager del JMF che gestisce i plugin, è possibile registrare un Demultiplexer. Se viene creato un Demultiplexer che implementa l'interfaccia com.advanced.media.IDemuxDynamicSource è possibile sfruttare la funzionalità di cambio sorgente dinamico con diversi tipi di DataSource.
Dal codice si nota lo sgancio del vecchio DataSource e l'aggancio del nuovo.

```
public void changeSource(DataSource source)
  throws IOException, IncompatibleSourceException
  {
    //Se non è stato inizializzato un datasource non è possibile
sostituirlo
    if (tracks == null)
      return;
      PushBufferStream[] copyPbs;
      /*
       * Viene tenuto l'elenco degli stream in copyPbs e solo dopo le
       * verifiche
       * viene copiato in streams
       * se viene lanciata qualche eccezione rimane attivo il
       * DataSource precedente
       **/
```

```
if (!(source instanceof PushBufferDataSource)) {
    throw new IncompatibleSourceException("DataSource not
    supported: " +              source);
} else {
  copyPbs = ((PushBufferDataSource) source).getStreams();
}

if ( copyPbs == null) {
    throw new IOException("Got a null stream from the
    DataSource");
}

if (copyPbs.length == 0) {
    throw new IOException("Got a empty stream array from the
    DataSource");
}

if (!supports(copyPbs))
    throw new IncompatibleSourceException("DataSource not
    supported: " +              source);


if ((copyPbs.length!=tracks.length)/*||(true)*/)
    throw new IncompatibleSourceException("DataSource has
              differents streams: " + source);

for (int i = 0; i < copyPbs.length; i++) {
    // viene sostituito ogni stream presente
        ((FrameTrack)tracks[i]).changePushBufferStream(
             (PushBufferStream)copyPbs[i]);
    }

    //viene fermato il DataSource da sostituire
    this.source.stop();
    this.source = source;

    //viene avviato il nuovo DataSource
    this.source.start();
    streams = copyPbs;
    }
```

Cap. 4 Progetto

Questa tesi di laurea ha come obiettivo primario l'ottimizzazione del JMF ai fini della realizzazione di sistemi per l'erogazione di servizi multimediali.
Nel capitolo precedente abbiamo spiegato quali modifiche sono state effettuate al JMF per permettere la realizzazione del progetto. In questo capitolo esemplificheremo, con un test costruito appositamente, i benefici di questa libreria e un applicativo, per la presentazione.

Il codice è stato suddiviso in 2 principali blocchi; quello relativo alle sole modifiche del JMF, presente nel package com.advanced.media, e tutto l'altro codice.
Per il progetto, il test e l'applicativo sono cosi distribuiti:

- nel package "progetto.test" è presente la libreria che simula gli scenari e permette di effettuare delle stime e dei confronti tra una soluzione standard con il JMF e una soluzione appoggiata alla libreria del progetto. Questa libreria di test permette la simulazione su un solo computer o l'utilizzo con più computer per il confronto tra le soluzioni e fornisce informazioni temporali e di utilizzo della memoria.
- nel package "progetto.presentazione" è presente l'applicazione che mostra i benefici della funzionalità aggiunta
- un terzo package "progetto.common" contiene le Api per la semplificazione della gestione degli stream basati sul protocollo RTP e alcune interfacce per una migliore suddivisione delle librerie sopra indicate.

4.1 Strumenti utilizzati

Per la realizzazione delle librerie si è utilizzato JAVA2SE nella versione 1.4 e l'ultima versione del JMF versione 2.1.1e.
Per lo sviluppo è stato utilizzato l'ambiente Eclipse.
E' stata utilizzata una Webcam per la creazione di alcuni video e Virtual Dub (un software per la manipolazione audio e video) per le modifiche dello stesso video in vari formati.

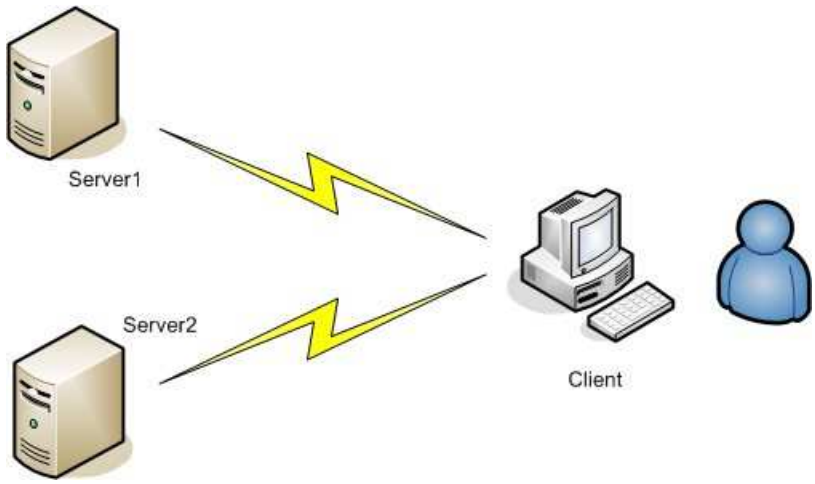
4.1.1 Creazione dei video utilizzati per i test

Con l'utilizzo di una webcam e il jmstudio è stato acquisito un video di circa 30 secondi con le dimensioni 176x144 e memorizzato con la codifica H263.
E' stato utilizzato il software VirtualDub per la realizzazione di altri video aventi stesso contenuto multimediale del video iniziale ma diversi effetti applicati con conseguente diversità nei colori. Questo per poter vedere la differenza delle librerie nei vari test.

4.2 Scenari di studio

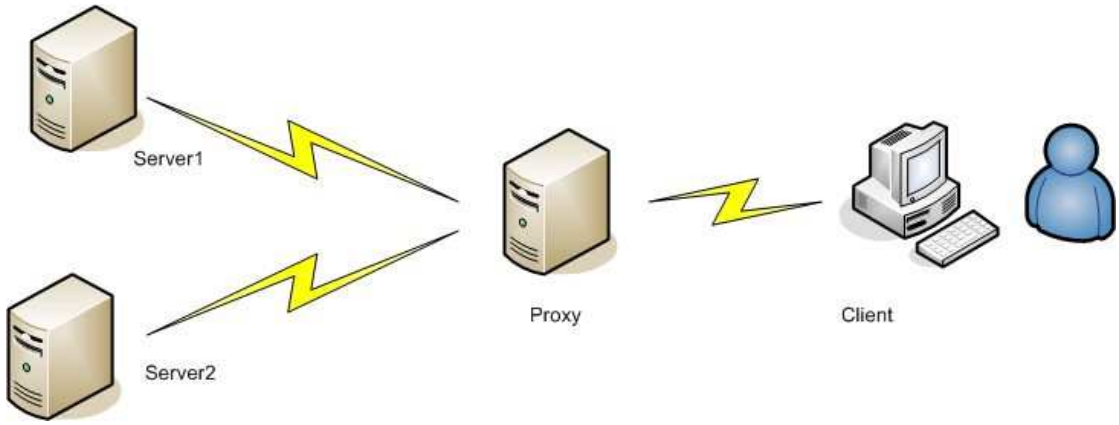
L'intero studio sul JMF è stato focalizzato sulla comprensione delle Api che entrano in gioco nel Client e nel Server in un generico servizio di streaming attraverso il protocollo rtp. Abbiamo due scenari particolari oggetto di studio rappresentati nelle figure sottostanti.

Figura 4.1 :scenario 1



Nel primo scenario abbiamo un cliente che usufruisce di un servizio di streaming (video on demand per esempio) dal Server1 e, per mantenimento della qualità del servizio (problemi di banda, di congestione o altro), è necessario che il flusso in streaming venga sostituito con un altro flusso spedito e gestito da un altro Server.

Figura 4.2 :scenario 2



Nel secondo scenario simuliamo una situazione analoga: il Client riceve il flusso video attraverso un Proxy. Il Proxy ha 2 Server registrati per la spedizione del flusso e uno dei due trasmette il flusso che verrà poi destinato al Client.
Per qualità del servizio si necessita il cambio di trasmissione dal Server al Proxy, perciò non appena il Proxy ritiene opportuno segnala al secondo Server di trasmettere .

4.3 Libreria di test per il monitoraggio delle risorse (package progetto.test).

Questa libreria è stata costruita per effettuare delle prove e qundi delle stime sulle differenze che intercorrono tra una soluzione del progetto con il JMF standard e una soluzione con il JMF modificato ad hoc.
Ci sono 3 tipi di differenze tutte legate tra loro, una che possiamo osservare facilmente e altre 2 che necessitano di apposite istruzioni inserite nella libreria:

- Una differenza visiva

  Questa differenza è importante perchè permette di osservare l'effetto negativo o meno che ha il Client
- Due differenze non visive; la memoria utilizzata e il tempo utilizzato

  Queste differenze non vengono sentite dal Client ma dal Proxy che vede o meno l'utilizzo ulteriore di risorse della macchina

Nei prossimi paragrafi spiegheremo le differenze di codice tra le due soluzioni, e il funzionamento della libreria per studiare i risultati ottenuti.

4.3.1 Soluzione con il JMF standard

Una soluzione basata sul JMF senza modifiche al framework lavora nel modo sotto descritto:
quando è necessaria la sostituzione del flusso multimediale, in entrambi gli scenari, viene svolto nuovamente tutti l'iter di prenotazione, allocazione e utilizzo delle risorse necessarie.
Questo porta ad un abbondante spreco di tempo e di risorse con conseguente degrado

della qualità del servizio.

Il codice riportato in figura è relativo al primo scenario e rappresenta la funzione (o metodo) importante; possiamo vedere come sia necessaria la chiusura del Player legato al vecchio flusso multimediale per impossibilità di riutilizzo e la creazione di un nuovo Player per il flusso aggiornato portando il sistema ad utilizzare maggiori risorse e ad interruzioni del servizio percettibili.

```
public void changeSource(DataSource ds){
  oldPlayer.stop();
  oldPlayer.close();
  Player newPlayer = Manager.CreatePlayer(ds);
  newPlayer.realize();
  while (newPlayer .getState()!=newPlayer.Realized);
  newPlayer.start();
}
```

Il codice relativo al secondo scenario (qui è interessato il Processor) mette in risalto il numero maggiore di operazioni necessarie per effettuare il cambio di flusso.

```
public void changeSource(DataSource ds)
{ oldProcessor.stop();
  oldProcessor.close();
  Processor newProcessor = Manager.CreateProcessor(ds);
  newProcessor.setSource(ds);
  newProcessor.configure();
  while (newProcessor .getState()!=newProcessor .Realized);
  newProcessor.setContentDescriptor(
  new ContentDescriptor(ContentDescriptor.RAW_RTP));
  newProcessor.realize();
  while (newProcessor .getState()!=newProcessor .Realized);
  DataSource dsForSink = newProcessor.getDataOutput();
  DataSink  Sender = Manager.createDataSink(dsForSink,destination);
  newProcessor.start();
  Sender.open();
  Sender.start();
}
```

4.3.2 Soluzione con il JMF modificato

La modifica effettuata al JMF elimina il bisogno di allocare nuove risorse relative al Player e riduce i tempi di sostituzione quasi a zero.
Questo perchè si modifica la situazione del Player in uso evitando attese temporali e spreco di risorse. Ogni cosa utilizzata è necessaria.

Il codice importante delle Api ad alto livello consta in poche righe all'interno del metodo changeSource() all'interno della classe Main dettagliata nei prossimi paragrafi.

Il codice relativo al primo scenario.

```
setStartTimeOp();
mp.changeSource(ds);
setStopTimeOp();
```

Il codice relativo al secondo scenario.

```
setStartTimeOp();
mproc.changeSource(ds);
setStopTimeOp();
```

4.4 Utilizzo della libreria per il test

La libreria per il test è stata creata al fine di effettuare si le stime ma anche di semplificare ed aiutare l'utente nei vari test su più macchine e su diversi sistemi operativi; cosa più importante permette la semplificazione degli scenari: è possibile effettuare ogni test utilizzando un solo computer.
Per poter osservare, in modo più approfondito, il consumo di risorse nello studio della libreria del progetto, si possono utilizzare anche strumenti di monitoraggio delle risorse legati al sistema operativo.
Dal test sarà possibile scegliere a piacimento quale dei due scenari simulare e osservare le differenze visive tra i 2 algoritmi usati.

Figura 4.3 : La figura mostra il pannello della libreria di test: dal menu si vede la possibilità di effettuare il test C-S o il test C-P-S. Tramite il pulsante Cambia Sorgente, l'utilizzatore è libero di provare più volte la funzionalità studiata e osservare gli differenze visive.

4.4.1 Dettagli libreria test

All'interno del package relativo alla libreria test (progetto.test) sono presenti quattro classi: *Main*, *Pannello*, *TestCS* e la classe *TestCPS*:

- La classe *Main* si occupa di salvare i risultati del test su file e di chiamare il pannello principale.
- La classe *Pannello*, si occupa della parte grafica della configurazione e dell'attivazione dei test. Offre la possibilità di disabilitare la simulazione del Server sulla stessa macchina e la possibilità di lavorare con la sola libreria del progetto senza passare dalla libreria "standard".
- Le classi TestCS e TestCPS rappresentano i due test le cui fasi saranno dettagliate in seguito.

Una volta avviato uno dei 2 test tramite il menu vengono registrate le attività in un frame apposito e l'utente è libero di utilizzare la libreria tramite un pulsante "Cambia sorgente" e osservare l'effetto sul Player che presenta il contenuto multimediale.
Le fasi del test Client Server sono le seguenti:

- la classe TestCS inizializza un oggetto ManagerRtpTransmitter (Api per la semplificazione delle trasmissioni con protocollo RTP) per la trasmissione dei flussi RTP simulando i Server;

- viene simulato un Client che, aggancia tramite l'oggetto ManagerRtpListener (Api per la ricezione dei flussi con protocollo RTP) i flussi e alloca il Player presentando il contenuto multimediale.
- a questo punto l'utente può cliccare sul pulsante "cambia sorgente" e osservare la diversità delle due librerie nell'attività in questione

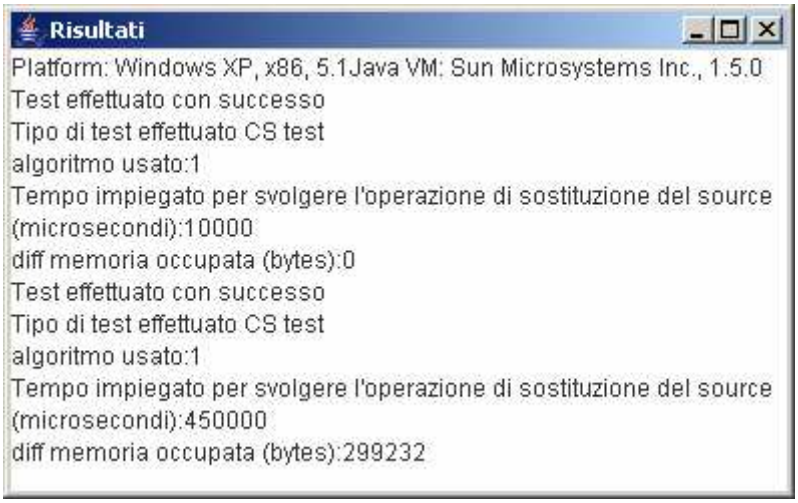
Le fasi del test Client Proxy Server sono più complesse:

- TestCPS avvia la trasmissione dei due flussi tramite l'oggetto ManagerRtpTransmitter simulando i Server.
- inizializza un oggetto Proxy che attende la ricezione dei due flussi tramite l'oggetto ManagerRtpListener e, a sua volta, tramite un Processor, spedisce il flusso al Client.
- Il flusso spedito dal Proxy viene agganciato dal Client simulato che crea il Player e presenta il filmato.

Al termine del test possiamo visualizzare i risultati dove possiamo osservare:

- tipo di macchina su cui stiamo lavorando;
- tipo di test effettuato
- tempi per effettuare il test
- memoria ocupata per il test

Figura 4.4 :Immagine contenente i risultati stampati dopo la conclusione del test. Si nota la piattaforma su cui è stato effettuato il test, il tipo di test effettuato e le risorse consumate.



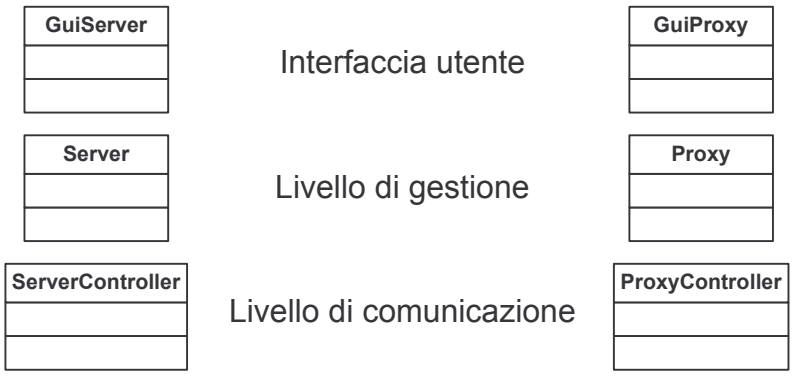
4.5 Applicazione di presentazione (package progetto.presentazione)

L'applicazione di presentazione è stata creata per mostrare le potenzialità della funzionalità aggiunta nel JMF; rappresenta un sistema distribuito per l'erogazione di un filmato. Questo sistema è composto da un *Proxy* e più *Server* per la rappresentazione del secondo scenario indicato nell'introduzione di questo capitolo. Il Client, come utilizzatore del servizio, è escluso .
Nei paragrafi successivi verranno mostrate le Api e il funzionamento dell'applicazione.

4.5.1 Approfondimento delle Api

Lo schema delle Api è molto semplice e l'immagine riportata sotto mostra la divisione dal livello più alto al livello più basso.

Figura 4.6 :Schema Api del Server e del Proxy



Dalla figura 4.6 si nota la suddivisione di entrambi i programmi in tre livelli:

- una parte relativa all'interfaccia utente
- una parte relativa al motore vero e proprio (Server , Proxy)
- una parte per la gestione dello streaming e la comunicazione tra i programmi

4.5.1.1 ServerController e ProxyController

La parte di gestione dello streaming vede due Api per la comunicazione tra il Proxy e il Server all'interno delle quali il Server si registra presso il Proxy e questo controlla e richiede la trasmissione del media al Server (Figura 4.6).

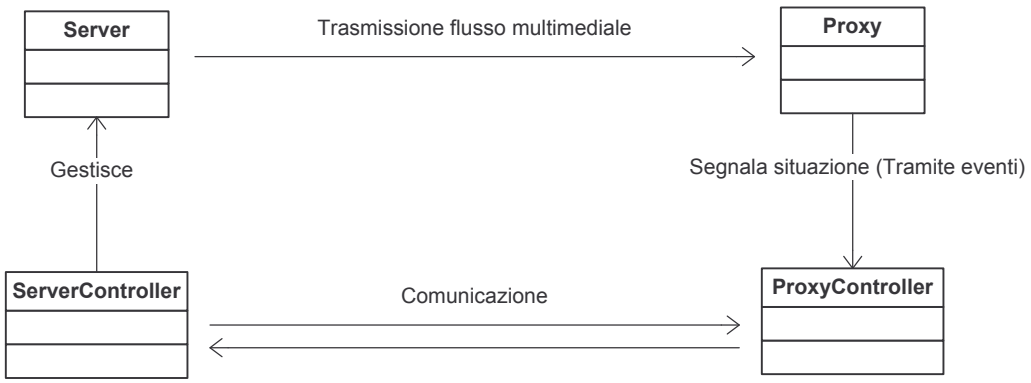


Figura 4.6 : La figura illustra i principi di funzionamento tra il Server e il Proxy. I due Controller comunicano. Il ServerController gestisce il Server in relazione ai comandi ricevuti dal ProxyController. Il Proxy tramite eventi segnala al ProxyController se si devono attivare altri Server alla trasmissione.

Questo controllo è basato sull'analisi continua dei pacchetti validi ricevuti da parte del Proxy: se i pacchetti invalidi, tra un ciclo di analisi e l'altro, superano il 30%, scatta una routine apposita nel Proxy, che segnala al ProxyController il problema; quest'ultimo segnalerà ad un secondo Server disponibile di iniziare la trasmissione del video.
Nei cicli di controllo immediatamente successivi si possono verificare due diverse situazioni:

- la situazione di ricezione del video si aggrava e viene superata la soglia del 50% di pacchetti invalidi. Scatta quindi una seconda routine nel Proxy che sostituisce lo stream attivo relativo al primo Server con lo stream del secondo Server;
- la situazione migliora e i pacchetti invalidi scendono sotto la soglia del 30%.
  Il Proxy segnala al secondo Server di fermare la trasmissione e si porta ad uno stato normale di trasmissione.

Se la situazione degenera rapidamente portando la percentuale di pacchetti invalidi da un valore sotto il 30% ad un valore sopra il 50%, a causa per esempio di un'interruzzione improvvisa della connessione, il Proxy inviterà tutti i Server disponibili alla trasmissione e non appena riceverà il video continuerà la fornitura del

servizio presso il Client.
Questò è il caso peggiore tra quelli possibili; si ha un interruzione del servizio della durata di un tempo che può variare da qualche millisecondo ad alcuni secondi in relazione allo stato dei Server disponibili.
Il fattore determinante in questa situazione è il tempo che impiega il flusso del video ad andare dal Server disponibile che lo trasmette presso il Proxy.

4.5.1.2 Server e Proxy

Per la realizzazione dei due applicativi, si sono create due Api per la semplificazione della ricezione e la trasmissione degli stream RTP: *ManagerRTPListenerAdvanced* e *StreamTransmitterRTPAdvanced*.
Grazie a queste abbiamo isolato il basso livello di configurazione del sessionManager per la ricezione, del Processor e del dataSink per la trasmissione.
Sopra a questi oggetti sono stati costruiti il Proxy e il Server.

4.5.1.4 Dettagli funzionamento Proxy

Il funzionamento del Proxy è leggermente più complesso:

- si imposta la porta di ascolta su cui ricevere i flussi multimediali con il metodo *addListeningPort*
- si configurano i parametri relativi alla destinazione con il metodo *configureDestinationParameter*
- si imposta il ProxyController
- si inizializza il Proxy alla ricezione dei flussi con il metodo initListening
- si prepara la trasmissione con il metodo prepareToTransmit
- con i metodi startTransmit e stopTransmit si regola la trasmissione del media

4.5.1.5 Dettagli funzionamento Server

Il funzionamento del Server si articola in pochi passi:

- si carica la configurazione dei parametri con la funzione *loadConfiguration*

si impostano i parametri internamente; sono il percorso dove trovare il media da spedire, il destinatario del media e la porta.

- si prepara il Server per la trasmissione con il metodo *prepare*
  all'interno di questo metodo vengono preparati oggetti interni che gestiscono la trasmissione del media.
- con i metodi startTransmit e stopTransmit si regola la trasmissione o meno del media

4.5.1.3 GuiServer e GuiProxy

Queste Api permettono la gestione grafica del Server e del Proxy .

Cap. 5 Analisi test e presentazione applicazione

In questo ultimo capitolo si effettuano le stime dei test svolti con la libreria apposita e si presenta la simulazione ricreata con l'applicazione di presentazione.

5.1 Test e valutazione dei risultati

Sono stati effettuati dei test su più computer di diversa potenza per poter valutare i benefici in termini del QOS della libreria utilizzata rispetto alla normale soluzione presentata .

I test effettuati sono relativi agli scenari presentati nel capitolo precedente; nel primo test abbiamo la seguente situazione:

- un Server spedisce un flusso video RTP e successivamente ne spedisce un secondo a sostituire il primo;
- un Client che presenta tramite Player il flusso video e alla richiesta di cambio sorgente effettua l'operazione registrando il tempo necessario per portare a termine la funzione e la quantità di memoria utilizzata.

Nel secondo scenario sono interessati:

- un Server che spedisce i flussi RTP;
- un Server Proxy che li riceve e spedisce un unico flusso al Client che usufruisce del servizio.
- Il Client che riceve il servizio e tramite il Player presenta il contenuto multimediale

5.1.1 Macchine su cui sono stati effettuati i test

I test sono stati effettuati su 2 macchine :

- Intel Pentium4 2Ghz 512 MB ram
- Athlon XP 1800 512 MB ram

E' stato effettuato un ciclo di test suddiviso sulle due macchine e un test "all in one" su ognuna delle macchine; sono perciò stati effettuati 6 cicli di test in totale (3 per il CS e 3 per il CPS).
Sono stati riportati i risultati in una tabella e successivamente sono state graficate le medie.

5.1.2 Risultati ottenuti

Per ogni ciclo di test si è ripetuto per dieci volte, nelle medesime condizioni, il test; nelle tabelle sottostanti riportiamo i consumi di tempo e memoria del sistema interessato.

I tempi sono indicati in millisecondi (ms) e la memoria occupata in KiloByte (KB)

Tabella 5.1 :Test CS suddiviso su due macchine (P4 Server – Athlon XP 1800 Proxy Client)

Tempi	Lib.Prog	0	0	0	1	0	0	1	0	0	0
Tempi	Lib. JMF	63	47	78	63	63	47	93	47	78	78
Memoria	Lib.Prog	2	0	2	0	0	2	0	0	0	0
Memoria	Lib. JMF	325	388	402	415	333	26	335	427	418	340

Tabella 5.2 :Test CPS suddiviso su due macchine (P4 Server – Athlon XP 1800 Proxy Client)

Tempi	Lib.Prog	0	0	0	0	16	0	0	0	0	0
Tempi	Lib. JMF	94	125	109	94	110	109	93	109	78	94
Memoria	Lib.Prog	0	2	2	0	2	0	0	0	0	0
Memoria	Lib. JMF	476	463	493	502	460	16	493	502	12	320

Tabella 5.3 :Test CS effettuato con la macchina Athlon XP (Server, Proxy, Client)

Tempi	Lib.Prog	0	0	0	0	0	0	0	0	0	0
Tempi	Lib. JMF	172	312	156	266	312	250	250	360	469	578
Memoria	Lib.Prog	0	0	0	0	0	2	0	0	0	0
Memoria	Lib. JMF	816	1433	244	111	274	284	-160	298	425	392

Tabella 5.4 :Test CPS effettuato con la macchina Athlon XP (Server, Proxy, Client)

Tempi	Lib.Prog	0	0	0	0	79	0	0	0	52	0
Tempi	Lib. JMF	890	500	249	797	469	188	141	172	356	652
Memoria	Lib.Prog	0	0	0	0	0	0	0	0	0	0
Memoria	Lib. JMF	505	72	820	-340	853	805	790	-263	18	550

Tabella 5.5 :Test CS effettuato con la macchina P4 (Server, Proxy, Client)

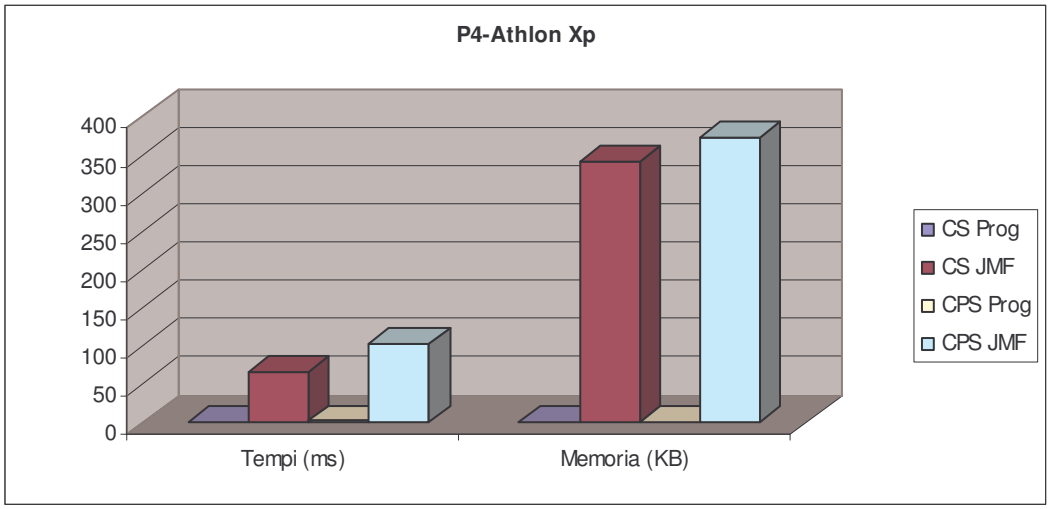
Tempi	Lib.Prog	1	0	0	0	0	0	13	0	0	0
Tempi	Lib. JMF	387	431	170	170	110	511	231	300	311	411
Memoria	Lib.Prog	0	0	0	0	0	0	2	0	0	0
Memoria	Lib. JMF	1100	339	841	873	342	358	358	359	-227	351

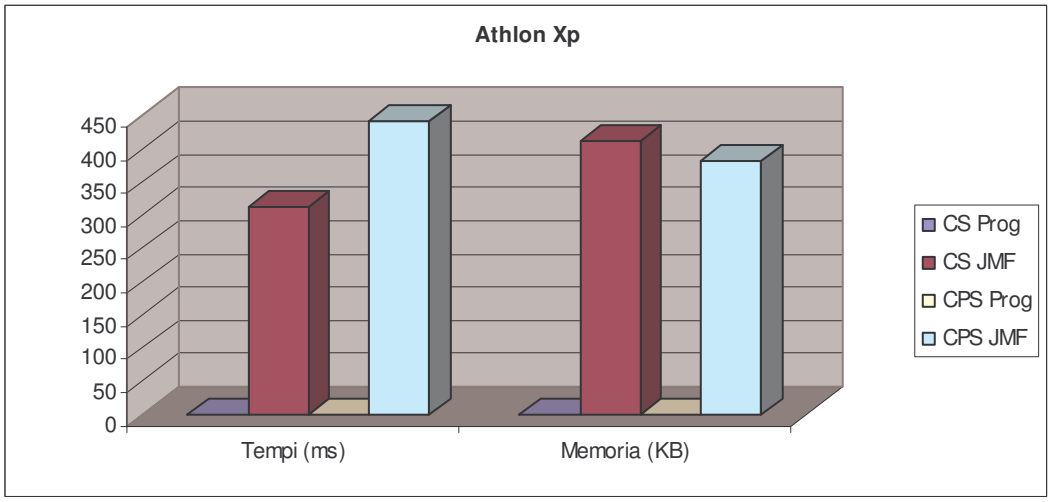
Tabella 5.6 :Test CPS effettuato con la macchina P4 (Server, Proxy, Client)

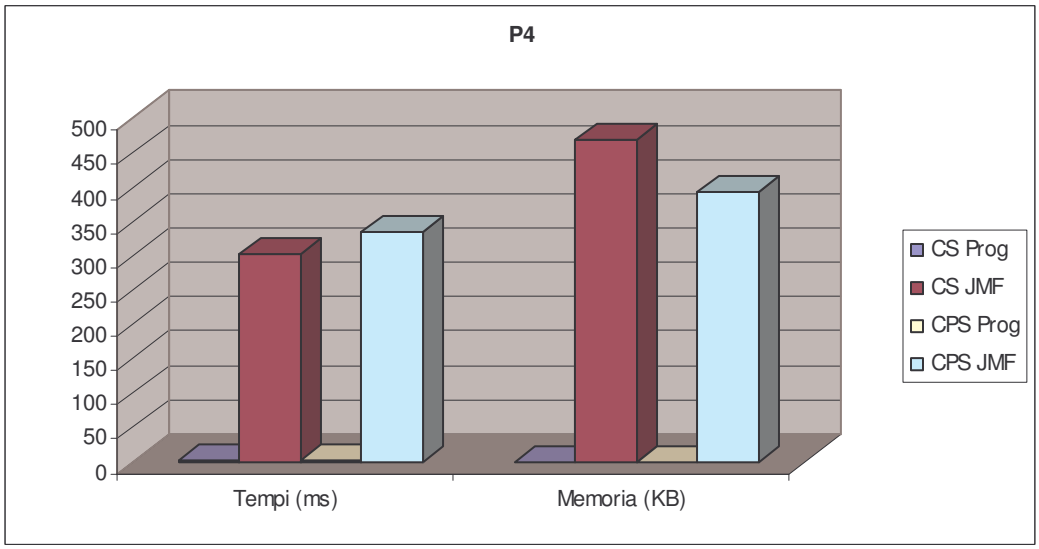
Tempi	Lib.Prog	0	0	0	25	0	0	0	0	1	0
Tempi	Lib. JMF	130	201	220	160	301	631	230	821	391	261
Memoria	Lib.Prog	2	0	0	0	0	0	0	0	0	0
Memoria	Lib. JMF	795	642	335	334	357	339	317	153	374	281

5.1.3 Riassunto test

Figure 5.7 , 5.8 , 5.9: Grafici che rappresentano le medie dei test scritti nelle tabelle precedenti.







5.1.4 Considerazioni sui risultati

Dai risultati si nota come il consumo di risorse e il tempo di attesa siano pressoche nulli segno della qualità delle modifiche al JMF. Si nota, d'altro canto, come sia lampante lo spreco di risorse senza le modifiche al JMF; spreco che interessa anche il Client visto che senza modifiche al framework la soluzione richiede che anche quest'ultimo riallochi degli oggetti (Player in particolare) per lavorare con un altro flusso multimediale.

5.2 Presentazione applicazione

Con la libreria di presentazione si è simulato il funzionamento di un'architettura per l'erogazione di un video . Durante il funzionamento, si sono creati volontariamente dei problemi alla connessione tra il Server e il Proxy osservando il comportamento del sistema. Non sono state fornite informazioni sui tempi o sui consumi perchè sono sufficienti i risultati ottenuti dalla libreria di test considerando che le simulazioni degli scenari sono le stesse.

5.2.1 Strumenti utilizzati

Si sono utilizzati 5 computer tutti collegati in rete etherenet a 100Mbit tra loro rappresentanti rispettivamente 1 Client, 1 Proxy e 3 Server:

- Il Client per la visione del filmato ricevuto
- Il Proxy come intermediario tra il Server e il Client
- I Server per la trasmissione del filmato al Proxy

5.2.2 Fasi della simulazione

Le fasi della simulazione sono 5:

- normale trasmissione Server Proxy Client;
- interruzione trasmissione Server Proxy;
- inizio trasmissione dei Server in attesa e recupero del servizio da parte del Proxy;
- interruzione trasmissione Server attivo;
- ulteriore cambio Server e recupero del servizio da parte del Proxy.

5.2.2.1 Dettagli della simulazione

Si è avviato il Client in attesa di ricezione del servizio. Si è configurato e avviato il Proxy. Si sono avviati i tre Server di cui uno ha iniziato a fornire il video al Proxy mentre gli altri due Server sono rimasti in attesa. Il Proxy ha quindi cominciato a fornire il servizio al Client trasmettendogli il video. Ad un momento non prestabilito si è scollegato il Server interrompendo la fornitura del servizio al Proxy. A quel punto il Proxy, accortosi dell'interruzione di trasmissione del flusso, ha attivato la procedura per

il cambio di fornitore del flusso; procedura che ha richiamato i Server disponibili alla trasmissione. Non appena il Proxy ha ricevuto un flusso dagli altri Server ha automaticamente sostituito il flusso entrante non più disponibile con uno disponibile continuando così a trasmettere al Client il filmato.

Si è nuovamente interroto il collegamento con il Server attivo in quel momento osservando che il Proxy ha automaticamente avviato la routine di sostituzione del flusso con il terzo Server. Il Client ha vissuto due brevi interruzzioni del servizio ma senza nessuna conseguenza in termini di consumo delle risorse sulla sua macchina.

Si è cosi mostrato come un sistema multipath per l'erogazione di servizi multimediali, è in grado di affrontare uno tra i peggiori problemi che si possono presentare nella rete internet; l'improvvisa interruzione del servizio da parte del Server.

Conclusioni

Nel corso del lavoro di tesi abbiamo avuto modo di studiare approfonditamente un framework per lo sviluppo di applicationi multimediali in Java, il JMF. Ne abbiamo compreso pregi e limiti strutturali tanto più evidenti quando si voglia utilizzare il JMF per fornire servizi multimediali avanzati e con qualità di servizio come quelli che prevedono una trasmissione su diversi percorsi (multipath) del materiale multimediale. Nello specifico abbiamo fornito una soluzione al problema del cambio della sorgente di un flusso multimediale a tempo di esecuzione, in modo da renderlo trasparente ai fruitori della sorgente dati stessa.

Tra le modifiche possibili del JMF abbiamo scelto quella più opportuna, ossia, in mancanza del codice sorgente relativo alla gestione del flusso RTP abbiamo aggiunto nel Demultiplexer la logica necessaria a fermare il DataSource in uso, e sostituirlo con un altro DataSource. Con la libreria sviluppata abbiamo poi realizzato un'applicazione per l'erogazione di flussi multimediali con modalità multipath, volta a verificarne l'usabilità e l'overhead introdotti.

I test effettuati hanno dimostrato l'efficacia e l'efficienza della modifica apportata al JMF. In particolare, abbiamo verificato che una soluzione senza modifiche al JMF non sarebbe praticabile, sia per la mancanza di API adeguate, sia per l'eccessivo spreco di risorse cui si andrebbe incontro. La soluzione proposta invece offre al livello applicativo API adeguate per il cambio della sorgente dati e non comporta spreco di risorse né in termini di tempo di CPU, né in termini di memoria utilizzata.

Riteniamo che la libreria sviluppata potrà essere utilmente impiegata nello sviluppo di servizi multimediali orientati alla qualità del servizio sviluppati in Java. In particolare sarà interessante estendere l'applicazione costruita nel progetto integrandola con librerie per il monitoring distribuito delle risorse di sistema in modo che il sottosistema di monitoring attivi la riconfigurazione automatica del percorso di servizio a fronte di guasti e/o congestioni nella rete.

Riferimenti bibliografici

[HOR01] Cay S. Horstmann, “Concetti di informatica e fondamenti di Java2”

[JAW02] Jamie Jaworski, “Java 2 tutto e oltre”

[SUN01] Sun Microsystem, Java Media Framework Guide 20 agosto 2003

[SUN02] Sun Microsystem, Java Media Framework Api documentation

[BOL01] Raffaele Bolla, Dispense “Servivi Multimediali” 2002/2003