UNIVERSITÀ DEGLI STUDI DI BOLOGNA
FACOLTÀ DI INGEGNERIA

Corso di Laurea in Ingegneria Informatica

Reti di Calcolatori L-A

Strumenti e Protocolli per il controllo dello Streaming

Tesi di laurea di:
Emanuele Gruppioni

Relatore
Chiar.mo Prof. Ing. Antonio Corradi
Correlatore
Ing. Luca Foschini

Anno Accademico 2003-2004

Parole chiave: **Erogazione di servizi multimediali**
Streaming
Protocolli di trasporto e di controllo
RTSP, Real Time Streaming Protocol
Java Media Framework

INDICE

Introduzione

Negli ultimi anni abbiamo assistito ad una veloce e rapida diffusione delle tecnologie legate ad Internet. In particolare l'erogazione di servizi multimediali attraverso la rete rappresenta una delle tecnologie più importanti e sicuramente con maggiore possibilità di sviluppo. In questo campo una buona comunicazione tra richiedente (client) e distributore (server) è basilare per l'erogazione del servizio. Si sono così sviluppate molte applicazioni di servizi multimediali con architetture di tipo proprietario che non permettevano, però l'interoperabilità con altri sistemi simili.

Un problema forte in questo ambito applicativo è quello dell'interoperabilità tra applicazioni non create appositamente per comunicare tra loro. Attraverso l'utilizzo di alcuni protocolli per la comunicazione in grado di gestire il trasporto ed il controllo dei dati, che si stanno imponendo come standard di riferimento, si può cercare di ottenere un architettura che garantisca una buona comunicazione fra il client e il server.

In questo scenario si pone questa tesi, con la quale si vuole studiare e costruire un applicazione che sia in grado di connettersi ad un server che supporti adeguatamente diversi protocolli considerati. Per realizzare l'applicazione verrà analizzato ed utilizzato un framework scritto in Java per la creazione di software relativo all'erogazione di servizi multimediali: Java Media Framework (JMF).

La tesi è organizzata come segue. Un primo capitolo in cui faremo una breve introduzione sulla distribuzione di servizi multimediali attraverso lo *streaming*. Nel secondo capitolo vengono presentati i protocolli per la comunicazione e nel terzo vengono messe maggiormente in rilievo le caratteristiche del protocollo di controllo (RTSP) che è alla base della comunicazione fra client e server. Nel quarto capitolo viene presentata la Java Media Framework introducendo gli aspetti legati al protocollo RTSP. Nel quinto capitolo vengono presentati gli strumenti e le tecnologie utilizzate per lo studio e la realizzazione dell'applicazione e nel sesto ed ultimo capitolo viene illustrata l'analisi sulle classi del JMF e la realizzazione pratica dell'applicazione con il corredo di test.

1 Streaming

1.1 Erogazione di servizi multimediali

Lo sviluppo tecnologico ha permesso di utilizzare Internet per veicolare contenuti come audio e video da sempre considerati “pesanti” per le dimensioni dei relativi file. Oggi non solo è possibile scaricare completamente questi file prima di ascoltarli o vederli, ma è anche possibile usufruirne durante il download (modalità *streaming*).

Soffermiamoci su questo punto per comprendere la differenza tra queste due possibilità.

La prima consiste semplicemente nel reperire il file audio o video che interessa e scaricarlo; una volta terminato il download lo si può ascoltare o vedere.

In questo caso prima di poter utilizzare il contenuto passa un certo periodo di tempo che dipende dalle dimensioni del file e dalla velocità della connessione.

Lo *streaming* invece, permette di usufruire del file man mano che questo viene scaricato (anche se non rimane un copia permanente sul disco) abbattendo così i tempi di attesa infatti non appena una piccola percentuale del file è caricata, l’apposito lettore inizia la riproduzione. In alcuni casi questa possibilità è fondamentale visto che alcune tecnologie ed applicazioni, senza di essa, non avrebbero senso. Un esempio per tutti è rappresentato dalle *Web radio*, le stazioni radio che consentono di ascoltare musica e altri contenuti su Internet in tempo reale. Infatti in questi casi non esiste un vero e proprio file ma piuttosto un flusso continuo (stream) di bit che vengono prodotti codificando in tempo reale la sorgente analogica (eventi *dal vivo*).

In ogni caso, lo *streaming* rappresenta un miglioramento rispetto al download per via del minore tempo di attesa necessario, anche nel caso in cui i tempi di attesa sono relativamente brevi.

Un altro vantaggio derivante dallo *streaming* sta nel fatto che quando si scarica un file non si conosce il suo esatto contenuto, per cui in certi casi, il file scaricato potrebbe non essere quello cercato mentre utilizzando lo *streaming* si può conoscere dopo una breve attesa quale sia il contenuto dello stream. In questo modo se il file non è quello cercato possiamo passare ad un altro stream (come se si cambiasse canale alla televisione) senza dover attendere del tempo per scaricare un altro file.

1.2 Streaming Server e Client

Per poter realizzare una trasmissione di dati multimediali in tempo reale attraverso Internet o su una rete locale occorrono almeno due applicazioni:

- un *server* (invia il file multimediale su Internet)
- un *client* (riceve il file multimediale)

Il server trasmette contenuti audio-video in risposta a richieste dei client. Le richieste vengono gestite utilizzando un protocollo per lo streaming (e.g. RTSP Real Time Streaming Protocol) mentre in generale il contenuto multimediale viene inviato utilizzando un protocollo per la trasmissione di dati attraverso la rete (e.g. RTP Real Time Transport Protocol).

1.3 Tipologia di contenuti

Il tipo di contenuti multimediali che si possono effettuare tramite lo streaming sono di due differenti tipologie:

- live
- on-demand

Gli eventi live possono essere concerti, dibattiti, convegni o altro ancora e vengono comunemente inviati attraverso Internet “in diretta” codificando in tempo reale la sorgente analogica ed utilizzando un software di broadcasting. In questo caso il server viene utilizzato per “riflettere” lo stream ai client. Indifferentemente dal momento in cui un client si collega, ognuno inizia a vedere o ascoltare l’evento dall’istante in cui si collega e non dall’inizio dell’evento. L’evento live può essere creato anche per contenuti multimediali già archiviati.

I client che richiedono (on-demand) un particolare file iniziano l’esecuzione dall’inizio essendo questo contenuto multimediale archiviato nel server .

1.4 Metodi di distribuzione

Multicast:

Un singolo stream viene condiviso dai client (vedi figura 1.1). Ogni client si collega ad un gruppo di multicast e riceve lo stream nello stesso modo in cui una radio si sintonizza per ricevere una normale trasmissione radio. Questa tecnica riduce la congestione di rete perché il server invia una sola copia dello stream attraverso la rete e le copie vengono create solo dove i collegamenti ai client si dividono. Questa tecnica non è molto diffusa per la distribuzione dei dati multimediali attraverso Internet, mentre viene utilizzata spesso per le reti locali.

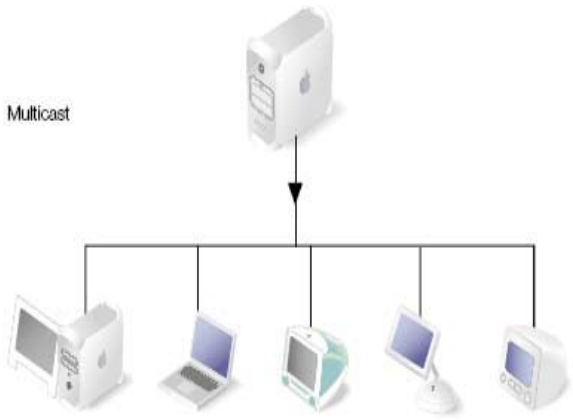

Figura 1.1

Unicast:

Ogni client richiede e riceve uno specifico stream generando molte connessioni punto a punto tra il server e i client (vedi figura 1.2). Molti client connessi in questo modo in una rete locale possono generare un intenso traffico di rete, ma questa tecnica rimane comunque la più utilizzato per la consegna di dati attraverso Internet.

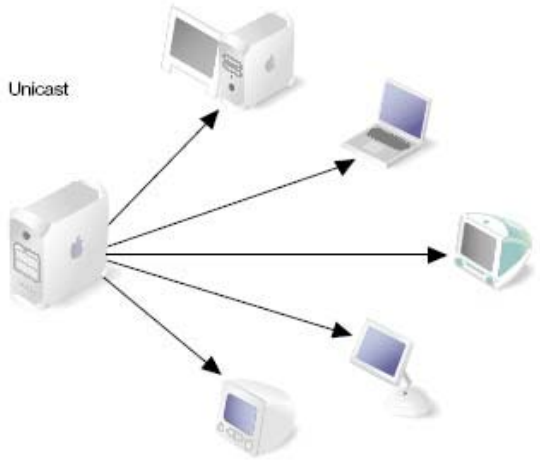

Figura 1.2

1.5 Sguardo sul funzionamento

Le trasmissioni radio o televisive sono spesso non compresse e consumano molta banda tra quella disponibile per la trasmissione. Questo, però, non è un problema infatti non sono in relazione con altre trasmissioni nella stessa frequenza attraverso la quale vengono inviate.
Invece quando si invia un contenuto multimediale attraverso Internet questo non ha una banda completamente dedicata allo streaming. In questo caso lo stream deve condividere una banda estremamente limitata con migliaia di altre trasmissioni che viaggiano in Internet.
Il contenuto multimediale inviato su Internet viene quindi codificato e compresso per la trasmissione. Il file così compresso viene salvato e il server lo invia attraverso Internet ai client.

2 Protocolli per lo Streaming

2.1 Introduzione

In questo capitolo ci proponiamo di introdurre alcuni protocolli che si stanno proponendo come standard nella distribuzione di contenuti multimediali attraverso la tecnologia di streaming.

Nella consegna di dati attraverso la rete si deve considerare che molti protocolli concorrono alla realizzazione del servizio e in particolare la gestione della sessione è una delle parti di maggior importanza nell'architettura di comunicazione multimediale. Si tratta infatti, della parte fondamentale per separare il controllo, di cui si ha bisogno durante il trasporto, dal trasporto stesso.

Lo streaming audio video necessita di vari protocolli per effettuare l'inizializzazione e il trasporto dei dati multimediali quelli più importanti sono:

- RTSP (Real Time Streaming Protocol) protocollo di controllo per la consegna dei dati
- RTP (Real Time Transport Protocol) protocollo di trasporto dei dati e RTCP (Real Time Control Protocol) protocollo pensato per lavorare in collaborazione con RTP e che effettua un ritorno sulla qualità della distribuzione
- RSVP (Resource ReSerVation Protocol) è un protocollo di controllo della rete che permette di riservare le risorse che vengono utilizzate per il trasporto dei dati
- SDP (Session Description Protocol) descrizione della sessione

Solitamente in aggiunta a questi protocolli per poter gestire la sessione multimediale si può utilizzare SIP (Session Initial Protocol) che è un protocollo per la creazione e la gestione delle conferenze multimediali.

Nella figura successiva vediamo quanto sia articolato e complesso il discorso relativo allo streaming e quanti soggetti può coinvolgere.

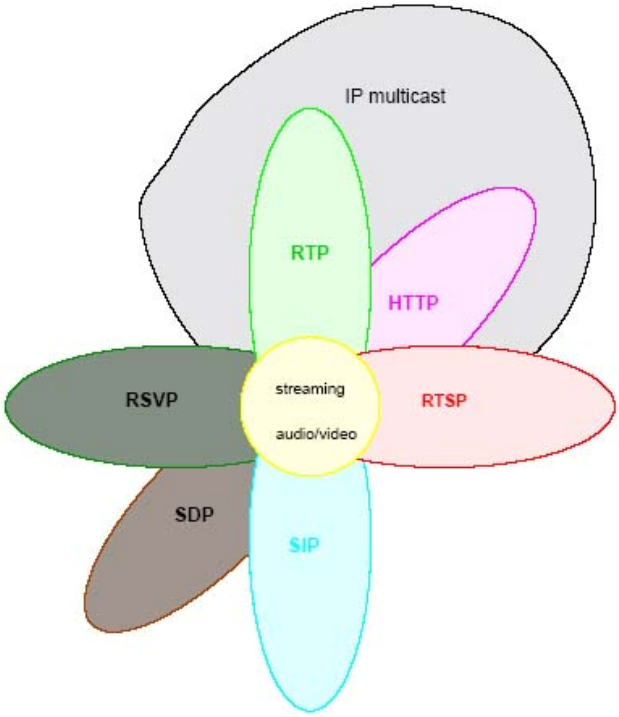

Figura 2.1 - Protocolli per lo Streaming

2.2 RTSP

Il protocollo RTSP è un protocollo di livello applicativo per il controllo della consegna di singoli o molteplici flussi di dati multimediali. Il suo scopo non è la consegna effettiva dei flussi di dati ma agisce come un "telecomando remoto" sui server multimediali.Il protocollo RTSP, infatti, è stato sviluppato con l'intenzione di fornire un insieme di comandi dedicati al controllo ed alla distribuzione di risorse multimediali attraverso la rete. I dati in questione possono essere semplici files audio o video oppure presentazioni, costituite da uno o più files differenti.I campi di applicazione potenziali sono moltissimi: Video on Demand, trasmissione di eventi Live, distribuzione di presentazioni multimediali, stazioni radio su internet o su Lan dedicate.
Il compito principale di RTSP è quello di governare il trasporto delle informazioni e non effettuare il trasporto delle informazioni stesse che viene realizzato da altri protocolli, come ad esempio RTP. Il meccanismo di trasporto prescelto per l'invio dei dati non influenza comunque le operazioni possibili con RTSP.

Il protocollo di RTSP è testuale e si basa sullo scambio di messaggi tra server e client; la sintassi è intenzionalmente simile a quella di HTTP in modo tale da fornire a RTSP gli stessi meccanismi di estensione di HTTP.

RTSP presenta tuttavia notevoli differenze con HTTP:

- RTSP introduce nuovi metodi ed ha un differente identificatore di protocollo
- Un server RTSP deve poter mantenere i suoi stati a seguito della ricezione o invio di comandi a differenza della natura *state-less* di HTTP.
- A differenza di HTTP sia il client che il server possono inviare dei comandi di richiesta.
- I dati vengono trasportati attraverso un protocollo ed un canale differente da RTSP.
- RTSP per definizione utilizza lo standard ISO 10646 (UTF - 8) piuttosto che ISO 8859 - 1 (HTML).
- L'URL di richiesta è sempre definita in maniera assoluta e non relativa rispetto ad un percorso o all'indirizzo del server.

Le operazioni supportate dal protocollo sono:

- **Controllo sul trasporto dei dati**: un client mediante comandi standard RTSP può gestire e controllare il trasporto dei dati, permettendo l'inizializzazione e la riproduzione di un contenuto multimediale
- **Recupero di informazioni da un media server**: un client può richiedere al server la descrizione di un file (anche attraverso HTTP o altri protocolli) cioè delle informazioni ad esso associate. Se il file in questione viene distribuito in multicast allora il server provvederà a fornire al client le informazioni relative all'indirizzo multicast e le porte associate per riceverlo. Se invece il file deve essere inviato in unicast al client allora sarà il client a fornire la porta e l'indirizzo di destinazione.
- **Invito di un server ad una conferenza**: un server può essere *invitato* a partecipare ad una conferenza o per fornire dei file oppure per registrare tutto o un sottoinsieme della conferenza stessa.
- **Aggiunta di un file ad una presentazione esistente**: un server può essere in grado di avvisare il client dell'esistenza di nuove risorse man mano che queste si rendono disponibili nel corso di una connessione.

Un analisi più approfondita di questo protocollo verrà effettuata nel capito seguente.

2.3 RTP e RTCP

2.3.1 RTP

RTP è un protocollo per la consegna punto a punto di dati multimediali che viene definito nell'RFC 1889 anche se non viene completamente specificato proprio perché RTP è stato progettato per essere "aggiustato su misura" ogni volta che lo si implementa. Esso è aperto a nuovi formati di payload e a nuovo software multimediale. Aggiungendo nuove specifiche sul profilo e sui payload, si può così adattare RTP ai nuovi formati dei dati e alle nuove applicazioni.

Internet è una rete a scambio di pacchetti, in cui i pacchetti inviati hanno un imprevedibile ritardo e jitter e i protocolli più comunemente utilizzati sono TCP e UDP. TCP fornisce un flusso tra due stazioni connesso e affidabile, mentre UDP da un servizio sulla rete non connesso e inaffidabile.

UDP è stato scelto come il protocollo di trasporto usato da RTP per due ragioni:

- RTP è in primo luogo pensato per i multicast mentre TCP , che è connesso, non scala bene e quindi non è appropriato
- In secondo luogo, per i dati in realtime, l' affidabilità non è così importante come la distribuzione tempestiva perchè anche in caso di perdita di pacchetti è necessario continuare la riproduzione dello stream

Inoltre, la trasmissione affidabile data dalle ritrasmissioni come in TCP non è desiderabile. Per esempio, in caso di congestione della rete, alcuni pacchetti possono andare persi e l' applicazione risulterebbe in una più bassa ma accettabile qualità. Se il protocollo insiste su una trasmissione affidabile, i pacchetti ritrasmessi potrebbero incrementare il ritardo, intasare la rete, ed eventualmente far morire l' applicazione ricevente.

Per questi motivi RTP è tipicamente incapsulato in UDP che offre inoltre funzioni di multiplexing e di controllo degli errori.

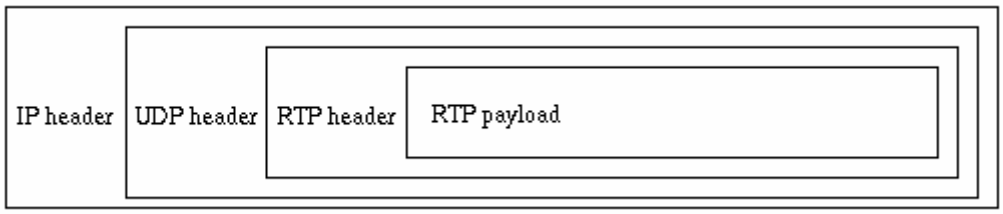

Figura 2.2 - Incapsulamento RTP Header

RTP viene visto nell'architettura come un estensione del protocollo di trasporto mentre dal punto di vista del software viene visto come parte vera e propria dell'applicazione per lasciare più controllo e flessibilità all'operazione di pacchettizzazione.
In sostanza RTP aggiunge ad UDP le seguenti funzionalità:

- Identificazione del *payload*
- Numeri di sequenza
- *timestamp*
- Capacità di identificare le sorgenti per la sincronizzazione

Timestamp è l' informazione più importante per le applicazioni *realtime*. Il mittente stabilisce il timestamp in funzione dell' istante in cui il primo byte nel pacchetto è stato campionato. Il timestamp viene incrementato del tempo coperto da un pacchetto. Dopo aver ricevuto i pacchetti dei dati, il destinatario usa il timestamp per ricostruire la sincronia originale al fine di mostrare i dati con il corretto ritmo. Il timestamp è anche usato per sincronizzare differenti stream con proprietà relative al tempo, come i dati audio e video nell' MPEG. Tuttavia, RTP in sè non è responsabile della sincronizzazione, che deve essere fatta al livello applicazione.
I numeri di sequenza sono usati per posizionare i pacchetti entranti nell' ordine corretto dato che UDP non porta pacchetti in ordine di tempo. Essi sono anche usati per la determinazione della perdita dei pacchetti. Si noti che nel formato video , quando un frame del video è diviso in più pacchetti RTP, tutti possono avere lo stesso timestamp. Così il solo timestamp non è sufficiente per mettere i pacchetti in ordine.
L' identificatore del payload specifica il formato del payload stesso così come gli schemi di codifica/compressione (vedi tabella 2.1). Da questo identificatore del payload, l' applicazione ricevente sa come interpretare e mostrare i payload.

PT	Name	Type	Clock rate (Hz)	Audio channels	References
0	PCMU	Audio	8000	1	RFC 3551
1	1016	Audio	8000	1	RFC 3551
2	G721	Audio	8000	1	RFC 3551
3	GSM	Audio	8000	1	RFC 3551
4	G723	Audio	8000	1	
5	DVI4	Audio	8000	1	RFC 3551
6	DVI4	Audio	16000	1	RFC 3551
7	LPC	Audio	8000	1	RFC 3551

8	PCMA	Audio	8000	1	RFC 3551
9	G722	Audio	8000	1	RFC 3551
10	L16	Audio	44100	2	RFC 3551
11	L16	Audio	44100	1	RFC 3551
12	QCELP	Audio	8000	1	
13	CN	Audio	8000	1	RFC 3389
14	MPA	Audio	90000		RFC 2250, RFC 3551
15	G728	Audio	8000	1	RFC 3551
16	DVI4	Audio	11025	1	
17	DVI4	Audio	22050	1	
18	G729	Audio	8000	1	
19	reserved	Audio			
20 - 24					
25	CellB	Video	90000		RFC 2029
26	JPEG	Video	90000		RFC 2435
27					
28	nv	Video	90000		RFC 3551
29 30					
31	H261	Video	90000		RFC 2032
32	MPV	Video	90000		RFC 2250
33	MP2T	Audio/Video	90000		RFC 2250
34	H263	Video	90000		
35 - 71					
72 - 76	reserved				RFC 3550
77 - 95					
96 - 127	dynamic				RFC 3551
dynamic	GSM-HR	Audio	8000	1	
dynamic	GSM-EFR	Audio	8000	1	
dynamic	L8	Audio	variable	variable	
dynamic	RED	Audio			
dynamic	VDVI	Audio	variable	1	
dynamic	BT656	Video	90000		
dynamic	H263-	Video	90000		

	1998				
dynamic	MP1S	Video	90000		
dynamic	MP2P	Video	90000		
dynamic	BMPEG	Video	90000		

Tabella 2.1 - Tipi di payload

Ad ogni dato tempo di trasmissione, un mittente RTP può soltanto mandare un solo tipo di payload, sebbene il tipo di payload possa cambiare durante la trasmissione, per esempio, per sistemare una congestione della rete.

Un'altra funzione è l' identificazione della sorgente che permette alle applicazioni riceventi di conoscere da dove il dato provenga. Per esempio, in una conferenza audio, dall'identificatore della sorgente un utente potrebbe dire chi sta parlando.

Per organizzare una sessione RTP, l' applicazione definisce una coppia particolare di endpoint di destinazione di trasporto (un indirizzo di rete più un paio di porte per RTP e RTCP). In una sessione multimediale, ogni mezzo è portato in una sessione RTP separata, con i propri pacchetti RTCP che riportano la qualità di ricezione per quella sessione. Per esempio audio e video viaggeranno su sessioni RTP separate, consentendo al destinatario di ricevere o meno un particolare media. Uno scenario di audio-conferenza presentato nell'RFC1889 illustra l' uso di RTP. Si suppone che ogni partecipante mandi dati audio in segmenti della durata di 20ms. Ogni segmento di dati audio è preceduto da un header RTP, e quindi il messaggio RTP risultante è posto in un pacchetto UDP. L' header RTP indica il tipo di codifica audio che si sta usando, ad esempio PCM. Gli utenti possono optare per cambiare la codifica durante una conferenza in reazione a una congestione della rete o, per esempio, per favorire le necessità di bassa ampiezza di banda di un nuovo partecipante alla conferenza. Le informazioni sul tempo e il numero di sequenza nell' header RTP sono usate dai destinatari per ricostruire la temporizzazione prodotta dalla sorgente, di modo che in questo esempio, i segmenti audio sono mostrati in modo contiguo al destinatario ogni 20 ms.

2.3.2 Campi fissi nell' header RTP

L' header RTP ha il seguente formato:

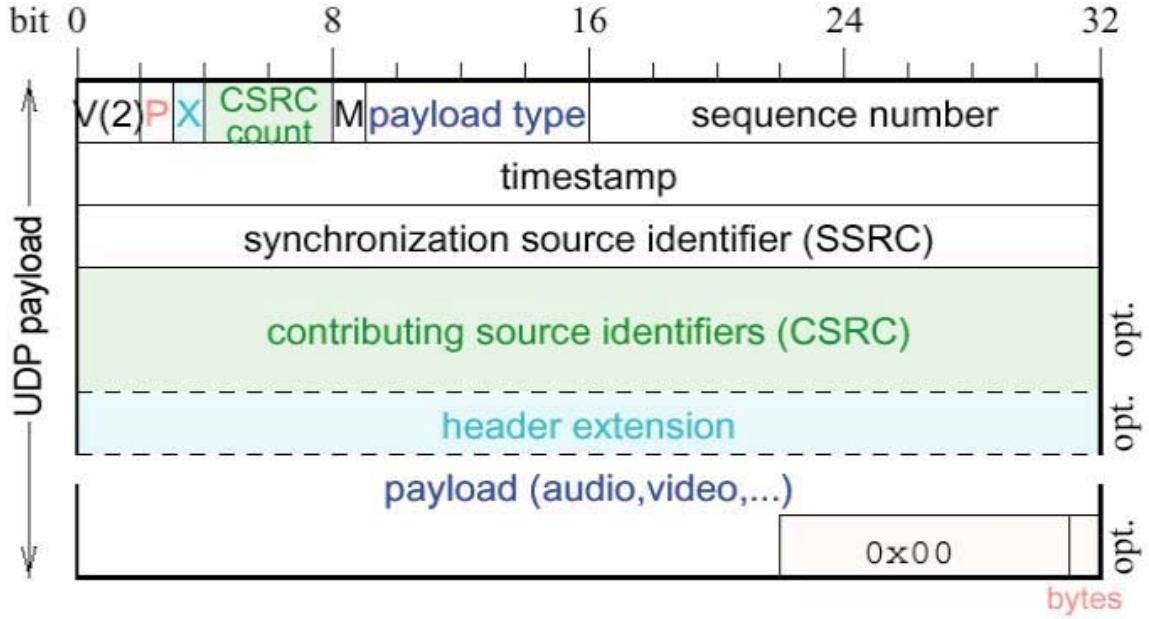

Figura 2.3 - Formato Pacchetto RTP

I primi dodici byte sono presenti in ogni pacchetto RTP, mentre la lista degli identificatori CSRC (sorgente fornita) è presente solo quando inserita da un mixer. I campi hanno il seguente significato:

Versione (V): 2 bit. Versione dell' RTP, l' ultima è la 2. (Il valore 1 è usato nella prima versione RTP e il valore 0 è usato dal protocollo inizialmente implementato nel VAT audio tool).

Padding (P): 1 bit se il bit padding è attivo, il pacchetto contiene alla fine uno o più padding bytes .(byte riempitivi addizionali, non facenti parte dei dati). L'ultimo byte padding contiene un valore di quanti byte padding dovrebbero essere ignorati. Questi particolari byte possono essere necessari per un'eventuale criptaggio con blocchi di misura fissa, o per trasportare pacchetti RTP in un'unità dati per protocolli di strati inferiori.

Estensione (X): 1 bit. Se settato, l' header fissato è seguito esattamente da un' estensione header.

Conteggio CSRC(CC): 4 bit. Il numero degli identificatori CSRC che seguono l' header fissato.

Marcatore (M): 1 bit. Definito da un profilo, il marcatore è inteso per permettere eventi significativi come limiti del frame da essere marcati in uno stream di pacchetti.

Payload (PT): 7 bit. Identifica il formato del payload RTP e determina la sua interpretazione dall' applicazione.

Numero di sequenza: 16 bit. Cresce di uno per ogni pacchetto dati RTP mandato, può essere usato dal destinatario per scoprire una perdita di pacchetti e per ristabilire la sequenza di pacchetti. Il valore iniziale è scelto casualmente per rendere difficili eventuali attacchi sul criptaggio.

Timestamp: 32 bit. L' istante preso a campione del primo byte nel pacchetto di dati RTP. Può essere usato per i calcoli di sincronizzazione e jitter. L'istante di campionamento deve essere derivato da un clock che s'incrementa monotonamente e linearmente nel tempo. Questo permette i controlli di sincronizzazione e le misure temporali sul campionamento dei pacchetti. Nel caso i pacchetti RTP siano generati periodicamente; bisognerà considerare l'istante di campionamento nominale, determinato dal clock di campionamento e non come lettura del clock di sistema. Il valore iniziale del timestamp è casuale, come per il sequence number. La frequenza di tale clock dipende dal tipo di codifica usata dal payload.

Molti pacchetti RTP consecutivi possono avere gli stessi timestamp se essi sono (logicamente) generati nello stesso istante. Pacchetti consecutivi possono contenere timestamp non monotoni se il dato non è trasmesso nell'ordine di campionamento (dipende dai tipi di compressione usati).

Synchronization Source SSRC:32 bit. La sorgente di uno stream di pacchetti RTP, è identificata da un identificatore SSRC, trasportato nell'header RTP in modo da non dipendere dall'indirizzo di rete. Tutti i pacchetti con la stessa synchronization source devono avere lo stesso clock e lo stesso generatore di sequence number, in modo che il ricevente raggruppi correttamente i pacchetti per il playback. I synchronization source possono essere riferiti a sorgenti di segnali come un microfono, una videocamera, o un mixer RTP (il mixer ha un suo SSRC il translator invece non è considerato una sorgente). L'SSRC è scelto in maniera random per essere probabilisticamente unico per una sessione RTP. Se un partecipante genera stream multipli in una sessione RTP, per esempio videocamera e microfono, ogni sorgente deve essere identificata con un SSRC differente.

Sebbene la probabilità che sorgenti multiple siano associate allo stesso SSRC è bassa, tutte le implementazioni RTP devono essere preparate ad evitare e risolvere le collisioni.

Lista delle Contributing Source CSRC: da 0 a 15 voci da 32 bit ognuna. Sono la lista degli SSRC che hanno contribuito a formare uno stream prodotto da un mixer. Questa lista è chiamata CSRC list. Di solito la funzione del mixer è proprio quella di condensare più flussi RTP, in un unico flusso. Il numero di CSRC è dato dal CC field. Se ci sono più di 15 contributing sources, soltanto 15 possono essere identificati.

Header extension: l'estensione dell'header è prevista solo per un uso limitato come ad esempio la sperimentazione. L'importante di fatto è saperla evitare per fare il parsing dei pacchetti correttamente. Tutte le informazioni su un'eventuale estensione sono a carico di un profile.

2.3.3 Interazioni tra RTP e RTSP

Gestione dei salti NPT(Normal Play Time) tramite RTP Media Layer

RTSP permette ai client di controllare sezioni non contigue di presentazioni, mostrando questi stream attraverso l'RTP media layer. I salti nel NPT (posizione assoluta del flusso di dati dall'inizio della presentazione) possono essere causati da operazioni di "spostamento" nel clip o da sequenze di PLAY PAUSE PLAY. L'RTP media layer che mostra gli stream non viene però influenzato da salti nell' NPT (vedi figura 2.4). Peraltro, sia le sequenze di numeri RTP che i timestamps devono essere continui e monotoni attorno ai salti di NPT.

Non si può assumere che il client RTSP possa comunicare con l' RTP media agent, dal momento che possono essere processi indipendenti. Se il timestamp RTP mostra lo stesso intervallo vuoto come fa l' NPT, il media agent, assumera' che c' e' una pausa nella presentazione. Se il salto in NPT e' lungo abbastanza, il timestamp RTP potrebbe "rovesciarsi" (roll over) e il media agent potrebbe credere che pacchetti in ritardo possano essere duplicati di pacchetti già visti.

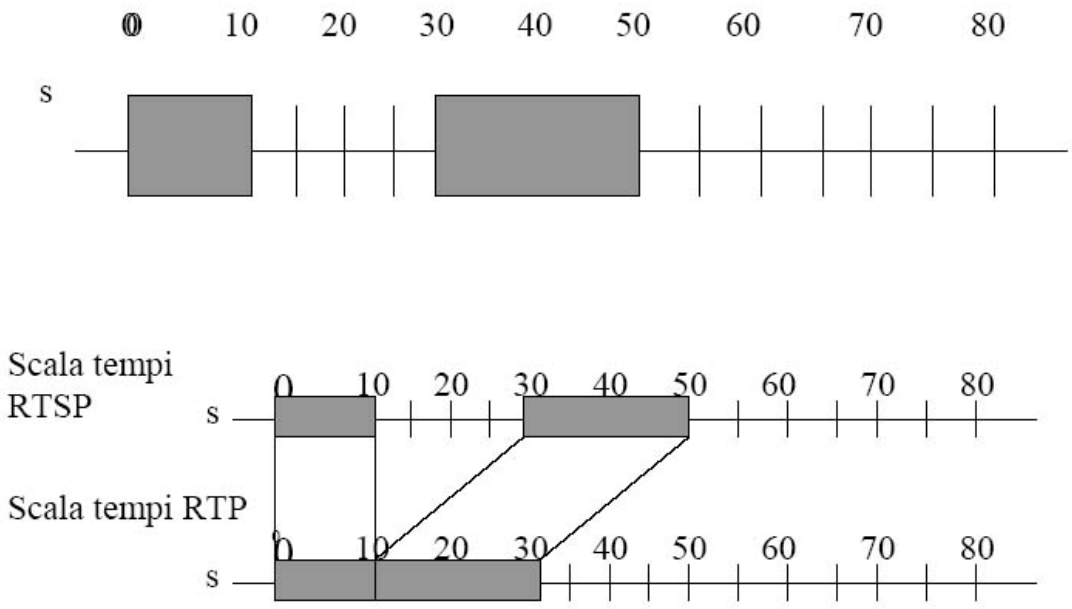

Figura 2.4 - Scala dei tempi RTSP RTP

Integrazione RTSP – RTP

Per particolari tipi di dati, sarà necessaria una stretta integrazione fra il livello RTSP e quello RTP. Client combinati RTP/RTSP dovranno usare il campo info di RTP per determinare se pacchetti RTP entranti siano stati mandati prima o dopo uno spostamento o prima o dopo un comando di PAUSE.

Mantenere una sincronizzazione NTP attraverso RTP timestamps

Il cliente puo' mantenere un corretto display dell' NPT per mezzo del valore del timestamp RTP del primo pacchetto dopo il riposizionamento. Il parametro sequence dell' header RTP-info fornisce il primo numero di sequenza del segmento successivo.

Audio continuo

Per l' audio continuo il server dovrebbe configurare il bit di controllo dell' RTP quando si inizia a servire una richiesta di PLAY. Ciò permette al client di svolgere un adattamento al ritardo in una rappresentazione. Per una corretta ripartizione dei tempi , i timestamp RTP dovrebbero corrispondere al tempo di playback.

2.3.4 RTCP

RTCP è il protocollo di controllo pensato per lavorare con RTP. Esso è standardizzato in RFC 1889 e 1890. In una sessione RTP, i partecipanti periodicamente mandano pacchetti RTCP per comunicare un ritorno sulla qualità della distribuzione dei dati ed informazione sull' appartenenza. RFC 1889 definisce cinque tipi di pacchetto RTCP per portare informazione sul controllo. Questi cinque tipi sono:

RR: rapporto sul destinatario. I rapporti sul destinatario sono generati dai partecipanti che non sono dei mittenti attivi. Essi contengono un ritorno della qualità della ricezione circa la distribuzione dei dati, incluso il numero più alto di pacchetti ricevuti, il numero di pacchetti persi, il jitter di inter-arrivo, e timestamp per calcolare il round-trip delay tra il mittente e il destinatario.

SR: rapporto sul mittente. I rapporti sul mittente sono generati da mittenti attivi. In aggiunta al feedback sulla qualità di ricezione come in RR, essi contengono una sezione d'informazione del mittente, fornendo informazioni sulla sincronizzazione fra vari media, su contatori cumulativi di pacchetto, e sul numero di byte mandati.

SDES: voci di descrizione della sorgente che contengono informazioni per descrivere le risorse.

BYE: indica la fine della partecipazione.

APP: funzioni specifiche dell' applicazione.

Attraverso questi pacchetti d' informazione di controllo, RTCP fornisce i seguenti servizi:

- **Monitoraggio della qualità del servizio e controllo di congestione**: Questa è la funzione primaria di RTCP. RTCP fornisce il feedback verso un' applicazione circa la qualità della distribuzione dei dati. L' informazione di controllo è utile ai mittenti, ai destinatari e ai monitor di terze parti. Il mittente può adeguare la sua trasmissione in base al feedback del rapporto del destinatario. I destinatari possono determinare se una congestione è locale, regionale o globale. Gli amministratori di rete possono valutare la performance della rete per la distribuzione multicast.
- **Identificazione della sorgente**: Nei pacchetti di dati RTP, le sorgenti sono identificate da identificatori a 32 bit generati casualmente. Questi identificatori non sono convenienti per gli utenti umani. I pacchetti RTCP SDES (descrizione della sorgente) contengono l' informazione testuale chiamata *nomi canonici* come

identificatori unici globali dei partecipanti alla sessione. Ciò può includere il nome dell' utente, il numero di telefono, l' indirizzo email ed altre informazioni.

- **Sincronizzazione fra i vari media**: I rapporti dei mittenti RTCP contengono un' indicazione del tempo reale e del corrispondente timestamp RTP. Questo può essere usato nella sincronizzazione fra vari media come la sincronizzazione fra voce ed immagini nei video.
- **Rappresentazione in scala dell' informazione di controllo**: I pacchetti RTCP sono mandati periodicamente fra i vari partecipanti. Quando il numero dei partecipanti cresce, è necessario equilibrare due possibili azioni: prendere un' informazione di controllo aggiornata, limitare il traffico di controllo. Per scalare fino a grandi gruppi mulicast, RTCP deve prevenire che il traffico di controllo inghiottisca le risorse di rete. RTP limita il traffico di controllo fino ad un massimo di 5% del traffico totale della sessione. Questa limitazione è rafforzata dall'adeguamento del tasso di generazione di RTCP in accordo con il numero di partecipanti.

2.4 SDP

Session Description Protocol (RFC2327) non è un vero e proprio protocollo, ma serve per descrivere i componenti delle sessioni multimediali a scopo di invito (SIP) o per altre forme di inizio sessione (RTSP,WEB).

La struttura della descrizione è suddivisa in tre sezioni:

- **sessione**: identifica e descrive i contenuti della sessione
- **tempistica**: informazioni temporali riguardanti l’inizio e la durata della sessione
- **trasmissione**: riguarda soltanto i media e contiene le informazioni necessarie per trasmettere e ricevere i flussi di dati

Descrizione della **sessione**:

v= (versione del protocollo)

o= (creatore del file e identificatore di sessione)

s= (nome della sessione)

i=* (informazioni sulla sessione)

u=* (descrizione del URI)

e=* (indirizzo e-mail)

p=* (numero di telefono)

c=* (informazioni sulla connessione non richieste se incluse in ogni singolo file multimediale)
b=* (informazioni sulla banda)

Informazioni **temporali**:
t= (tempo in cui la sessione è attiva)
r=* (ripetizioni temporali: settimanali, ecc.)
z=* (cambio di fuso orario)
k=* (chiave di codifica)
a=* (zero o più linee di attributi)

Descrizione sulla **trasmissione** del file multimediale:
m= (formato multimediale e indirizzo di trasporto)
i=* (titolo del media)
c=* (informazioni di connessione, opzionali se incluse a livello di sessione)
b=* (informazioni sulla banda)
k=* (chiave di codifica)
a=* (zero o più linee di attributi)

2.4.1 Uso di SDP per la descrizione di una sessione RTSP

Ora descriveremo come un file SDP, richiesto, per esempio, attraverso HTTP, determina la gestione di una sessione RTSP. Ciò inoltre descrive come un client debba interpretare un contenuto SDP ritornato da una risposta ad una richiesta di DESCRIBE. L' SDP non fornisce alcun meccanismo per il quale un client possa distinguere, senza la guida umana, fra diversi media di streaming da produrre simultaneamente e un set di alternative (ad esempio due stream audio parlati in lingue differenti).

URL di controllo

L' attributo "a=control:" è usato per comunicare l’URL di controllo. Questo attributo è usato sia per le descrizioni degli stream che quelle delle sessioni. Se usato per un media particolare, indica l’URL da usare per controllare quel particolare streaming di dati

multimediali. Se trovato al livello di sessione, l'attributo indica l'URL per il controllo aggregato.

Esempio:

a=control:RTSP://example.com

Le Implementazioni dovrebbero cercare per un URL base nel seguente ordine:

1. Il campo RTSP Content-Base
2. Il campo RTSP Content-Location
3. La richiesta di una URL RTSP

Se questo attributo contiene solo un asterisco (*), allora l'URL è trattata come se vi fosse un URL vuota inclusa, e quindi eredita l' intera URL di base.

Media stream

Il campo "m=" è usato per numerare gli stream. Ci si aspetta che tutti gli stream specificati siano mostrati con l' appropriata sincronizzazione. Se la sessione è unicast, il numero di porta serve come un consiglio dal server al client; il client deve includerla nella sua richiesta di **SETUP** e può ignorare questa raccomandazione. Se il server non ha preferenze dovrebbe settare la porta al valore zero.

Inoltre il campo "m" contiene informazioni riguardanti il tipo di protocollo, il profilo e possibili livelli sottostanti necessari per il trasporto dello stream.

Esempio:

m=audio 0 RTP/AVP 31

Payload

I "payload" sono specificati nel campo "m=". Nel caso in cui il "payload" sia un "payload" da RFC 3551, nessun altra informazione è richiesta. Nel caso sia un "payload" dinamico, l' attributo del media "rtpmap" è usato per specificare il tipo di contenuto multimediale. L'attributo "encoding name" all'interno di "rtpmap" può essere uno di quelli specificati in RFC 3551 o una codifica sperimentale come prefisso come specificato in SDP (RFC2327). I parametri propri della codifica non sono specificati in questo campo, ma piuttosto nell' attributo "fmtp" descritto successivamente.

Parametri specifici di formato

I parametri specifici di formato sono comunicati usando l' attributo del media "fmtp". La sintassi dell' attributo "fmtp" è specifica della codifica a cui l' attributo si riferisce. Nota che l' intervallo di pacchettizzazione è comunicato usando l' attributo "ptime".

Estensione della rappresentazione.

L' attributo "a=range" definisce il tempo totale dello stream o della sessione fornita. (La lunghezza delle sessioni live può essere dedotta dai parametri "t" e "r"). A meno che la presentazione contenga media stream di differenti durate, l' attributo lunghezza è un attributo del livello di sessione. L' unità è specificata prima, seguita dal valore dell' estensione. Le unità e i loro valori sono come definiti per NPT, SMPTE, Absolute time:

- **NPT (Normal Play Time)**: Indica la posizione assoluta del flusso di dati dall'inizio della presentazione (e.g. npt=123.45- 125).
- **SMPTE Relative Timestamp**: Indica la posizione dall'inizio del clip. Viene espresso tramite il codice SMPTE che fornisce il seguente formato: ore:minuti:secondi:frames.subframes (e.g. smpte=10:07:00-10:07:33:05.01).
- **Absolute time**: Si riferisce allo scorrere del tempo secondo lo standard ISO 8601 e si basa su UTC (Coordinated Universal Time) (e.g. 8 Novembre 1996 alle 14:37:20 e un quarto di secondo diviene: 19961108T14372025Z).

Esempi:

a=range:npt=0-34.4368

a=range:clock=19971113T2115-19971113T2203

Tempo di disponibilita'

Il campo "t=" deve contenere adeguati valori per i tempi di inizio e fine per entrambi i controlli aggregati e non aggregati dello stream. Il server deve indicare il valore per il tempo di stop per il quale garantisce che la descrizione sia valida e un tempo di inizio che sia uguale a o precedente il tempo al quale la richiesta di DESCRIBE è stata ricevuta. Ciò può anche indicare i tempi di inizio e fine dello 0, intendendo che la sessione sia sempre disponibile.

Per sessioni "dal vivo" i campi "t=" ed "r" possono essere usati per indicare l'inizio di un evento. Il tempo di fine può essere fornito ed indica la possibilità che la sessione "live" termini entro quel dato tempo.

Informazione sulla connessione

In SDP il campo "c=" contiene l' indirizzo destinazione per il media stream. Tuttavia, per gli stream unicast on-demand e alcuni stream multicast, l' indirizzo destinazione è specificato dal client attraverso la richiesta di SETUP. A meno che il contenuto del media abbia un indirizzo destinazione fisso, il campo "c=" deve essere portato ad un appropriato

valore nullo. Per gli indirizzi di tipo "IPv4", questo valore è "0.0.0.0" e per indirizzi di tipo “IPv6” il valore da impostare è “0.0.0.0.0.0”.

Entity tag

L' attributo opzionale "a=etag" identifica una versione di una descrizione di sessione ed è opaco al client. Le richieste di SETUP possono includere l' identificatore nel campo If-Match solo per permettere l' instaurazione di una sessione se questo valore corrisponde ancora a quello della corrente descrizione. Il valore dell' attributo è anch’esso opaco e può contenere qualsiasi carattere permesso tra i valori degli attributi SDP.

Esempio:

a=etag:158bb3e7c7fd62ce67f12b533f06b83a

Controllo aggregato non disponibile

Se una presentazione non supporta il controllo aggregato e le sezioni multiple di media sono specificate, ogni sezione deve avere l’URL di controllo specificata attraverso l'attributo "a=control:",mentre non esiste l’attributo “a=control:” del livello sessione.

Esempio:

v=0

o=- 2890844256 2890842807 IN IP4 204.34.34.32

s=I came from a web page

e=adm@example.com

c=IN IP4 0.0.0.0

t=0 0

m=video 8002 RTP/AVP 31

a=control:RTSP://audio.com/movie.aud

m=audio 8004 RTP/AVP 3

a=control:RTSP://video.com/movie.vid

Si noti che la posizione dell' URL di controllo nella descrizione implica che il client instauri sessioni separate di controllo RTSP verso i server audio.com e video.com. E' consigliato che un file SDP contenga la completa informazione dell' inizializzazione del contenuto multimediale anche se è demandata al client del media attraverso mezzi non-RTSP. Questo è necessario nel momento in cui non c'è alcun meccanismo per indicare che il client debba richiedere informazioni piu' dettagliate sul media stream attraverso il comando DESCRIBE.

Controllo aggregato disponibile

In questo caso, il server gestisce stream multipli che possono essere controllati come un unico. Quindi, ci sono entrambi gli attributi "a=control:" del livello del media, che sono usati per specificare l'URL dello stream, e un attributo "a=control:" del livello sessione che è usato come richiesta URL per il controllo aggregato.

Se la presentazione comprende solo uno stream singolo, l' attributo "a=control:" del livello media può essere omesso del tutto. Tuttavia, se la presentazione contiene più di uno stream, ogni sezione del media stream deve contenere il proprio attributo "a=control:".

Esempio:

C->S: DESCRIBE rtsp://example.com/movie RTSP/1.0

CSeq: 1

S->C: RTSP/1.0 200 OK

CSeq: 1

Date: 23 Jan 1997 15:35:06 GMT

Content-Type: application/sdp

Content-Base: rtsp://example.com/movie/

Content-Length: 164

v=0

o=- 2890844256 2890842807 IN IP4 204.34.34.32

s=I contain

i=<more info>

t=0 0

c=IN IP4 0.0.0.0

a=control:RTSP://example.com/movie/

m=video 8002 RTP/AVP 31

a=control:trackID=1

m=audio 8004 RTP/AVP 3

a=control:trackID=2

In questo esempio, il client è richiesto per stabilire una singola sessione RTSP con il server, e usa le URL:

RTSP://example.com/movie/trackID=1 e RTSP://example.com/movie/trackID=2 per organizzare gli stream audio e video, rispettivamente.
L' URL RTSP://example.com/movie/ controlla l' intero film

Trasporto di SDP al di fuori di RTSP

Inoltre si devono fare alcune considerazioni quando la descrizione della sessione viene consegnata al client senza l'utilizzo di RTSP, ma utilizzando HTTP o l'email:

- SPD deve contenere URL assoluti, perché in molti casi gli URL relativi non funzionano correttamente.
- la probabilità che le informazioni scritte nei campi di "disponibilità" siano valide è molto più alta se vengono rilasciate dopo un DESCRIBE, piuttosto che nei casi in cui vengano recuperate usato altri metodi

2.5 SIP

Il SIP (Session Initial Portocol) è un protocollo di segnalazione definito dalla IETF nella RFC 2543 del marzo 1999, poi sostituita dalla RFC 3261 del giugno 2002. Il suo scopo è stabilire chiamate e conferenze real-time su reti basate sull'Internet Protocol.
Il protocollo SIP, proprio perché definito come parte dell'architettura IETF, permette un'ottima integrazione con gli altri protocolli definiti per la reti IP, come RTP/RTCP, RTSP, RSVP, SDP, ma non risulta dipendente da essi e così può essere utilizzato senza problemi insieme ad altri protocolli di segnalazione. Il protocollo SIP consente inoltre la separazione dei dispositivi fisici, come il telefono, dagli utenti e la logica del server dal controllo centralizzato. La separazione del servizio dai dispositivi fisici permette l'implementazione di funzionalità quali la gestione di informazioni sulla presenza o meno di utenti ed il supporto della mobilità. Il protocollo usato nel trasporto dei messaggi di segnalazione è l'UDP, è stato scelto per poter ridurre i tempi di Call Set Up, in questo modo infatti riusciamo ad eliminare la componente del ritardo si set up relativa al tempo necessario per l'instaurazione della connessione TCP. Per poter però garantire alla trasmissione dei messaggi l'affidabilità a livello applicazione viene utilizzato un meccanismo di un time out ed il meccanismo di richiesta-risposta, tipico del protocollo.

In pratica se il chiamante non riceve una risposta alla sua richiesta entro un tempo prestabilito, il time out appunto, ne deduce che è andata persa e la ritrasmette. Il chiamato invece riceve un ACK dal chiamante quando a questi arriva la risposta. Se però, il chiamante registra per molte volte il fallimento della richiesta può decidere di aprire una connessione tramite il TCP.
Inoltre, a livello di sicurezza il SIP offre un meccanismo di autenticazione e di controllo che permette al software che gestisce le chiamate in arrivo di rifiutare eventuali tentativi di chiamate indesiderate.

2.5.1 Architettura SIP

Il SIP è un protocollo di tipo client-server ed i suoi principali componenti sono l'User Agent, il Proxy Server, il Redirect server ed il Registrar.
User Agent: o *SIP endpoint*, è un sofware, di PC o di terminale telefonico, che interagisce direttamente con un utilizzatore umano; funziona come *Client* (UAC) quando inizia una richiesta, come *Server* (UAS) quando vi risponde. Due UA possono comunicare tra loro direttamente o tramite server intermedi ed hanno inoltre la possibilità di immagazzinare e gestire lo stato delle chiamate.
Proxy Server: inoltra le richieste dall'UA al successivo SIP server o ad un altro UA interno alla rete, stabilendo così una politica di instradamento a livello di servizio di chiamata.
Redirect server: allevia la mole di lavoro del Proxy Server, ha il compito di rispondere alle richieste del *Client* e di aggiornalo sul indirizzo del server richiesto successivo. In questo modo il *Client* potrà inoltrare la richiesta successiva all'indirizzo corretto.
Registrar: è un server che accetta richieste di registrazione e provvede a memorizzare le informazioni lì contenute in un Location Server tramite un protocollo non-SIP.

2.5.2 Richieste e Risposte SIP

Il protocollo SIP, grazie al fatto che è basato sul modello client-server, risulta essere perfettamente integrato con Internet e la sua filosofia. La sintassi, sia delle richieste che delle risposte segue, infatti, le specifiche dei messaggi HTTP.

I messaggi sono caratterizzati da una *Start-Line*, da uno o più campi d'intestazione, da una linea vuota, che serve per indicare la fine dei campi d'intestazione, ed infine da un corpo messaggi opzionale.

Le richieste, o metodi, implementate nel protocollo sono le seguenti:

- INVITE: Inizia una chiamata invitando un utente a partecipare ad una sessione
- ACK: Conferma che l'UAC ha ricevuto una risposta finale ad una richiesta
- BYE: Indica il termine di una chiamata
- CANCEL: Cancella una richiesta pendente
- OPTIONS: Richiede le capacità disponibili ad un server
- REGISTER: Registra l'User Agent

Le risposte invece sono caratterizzate da uno *Status-Code*, cioè da un intero di tre cifre che indica il risultato del tentativo di capire e soddisfare la richiesta. La prima cifra definisce la classe della risposta, le altre non hanno un ruolo preciso. Sotto vengono illustrate le possibili classi di risposta:

- 1XX: *Provisional* indica che una richiesta è stata ricevuta, ma che il server contattato non ha ancora una risposta definitiva e sta continuando a processare la richiesta
- 2XX: *Success* indica che la richiesta ha avuto successo.
- 3XX: *Redirection* indica che si hanno nuove locazioni dell'utente e quindi sono necessarie altre azioni per poter completare la richiesta.
- 4XX: *Request Failure* indica che la richiesta contiene una sintassi errata o non può essere completata da questo server, deve essere quindi modificata ed in seguito si può ritentare.
- 5XX: *Server Failure* indica che il server non riesce a completare una richiesta apparentemente valida
- 6XX: *Global Failure* indica che la richiesta non può essere completata da alcun server.

Nei singoli messaggi di richiesta o di risposta sono inseriti diversi campi intestazione, di seguito sono illustrati i più interessanti:

- Call-ID: è l'unico identificatore globale della chiamata o delle registrazioni di un particolare *Client*.

- TO: contiene l'indirizzo logico del chiamato, può non essere quello definitivo visto che il chiamato può spostarsi nella rete.
- FROM: indica chi ha iniziato la richiesta e deve essere presente sia nelle richieste che nelle risposte.
- Contact: contiene una lista di indirizzi dove un utente può essere trovato per le richieste future o per ulteriori comunicazioni.
- Cseq: è un campo presente nelle richieste, contiene un numero intero decimale, scelto dal *Client* ed unico all'interno di un singolo valore di Call-Id. Contiene anche il metodo della richiesta.
- VIA: Indica il percorso fatto dalla richiesta fino ad un certo momento.In questo modo le risposte possono seguire lo stesso percorso delle richieste.
- Content-type: Indica il protocollo usato nel corpo del messaggio. Di solito si tratta di SDP.
- Exipres: indica il tempo di validità di una registrazione.

2.5.3 Instaurazione di una sessione SIP

Per stabilire e terminare sessioni multimediali, per sessioni s'intendono scambi di voce, video tra più partecipanti, il protocollo SIP implementa cinque servizi:

- **User Location**: serve per determinare il terminale da utilizzare per la comunicazione
- **User capabilities**: serve per determinare i media e i parametri dei media da utilizzare.
- **User availability**: serve per determinare la disponibilità del chiamato a iniziare una comunicazione.
- **Session Setup**: "ringing", instaurazione dei parametri di sessione entrambe le chiamante e chiamato.
- **Session Menagement**: include il trasferimento e la terminazione di una sessione, la modifica dei parametri e la richiesta di determinati servizi.

La procedura per instaurare una sessione consiste nell'invio da parte del chiamante di una richiesta INVITE, per notificare al chiamato la sua volontà di aprire la sessione, il chiamato può accettare la richiesta con un messaggio di risposta positivo (2xx), la

conferma della ricezione viene comunicata tramite un messaggio ACK. Lo scambio di questi messaggi può avvenire in due modalità:
Proxy server: in questo caso è lo stesso Proxy a provvedere ad inoltrare le richieste direttamente al chiamato o al proxy successivo.
Redirect server: in questo caso il redirect fornisce al chiamante le informazioni necessarie perché sia lui stesso ad inviare le richieste al chiamato.

2.6 Relazione tra protocolli

Per effettuare lo streaming audio video su Internet i protocolli appena descritti devono interagire tra loro creando così un'unica architettura per l'inizializzazione la gestione e il trasporto dei contenuti multimediali (vedi figura 2.5).
L'inizializzazione della sessione può avvenire per mezzo della richiesta RTSP di DESCRIBE da parte del client che richiede un particolare contenuto multimediale. Il server risponde inviando una risposta oltre all'accettazione della richiesta una descrizione SDP del file che vogliamo ottenere, nella quale vengono descritte le caratteristiche delle tracce audio video del contenuto richiesto. Il client invia poi un comando di SETUP per definire il tipo di trasporto da utilizzare comunicando al server anche le porte alle quali inviare i dati. La risposta di conferma del server contiene tutte le informazioni per l'inizializzazione del client e inoltre il numero della sessione multimediale che non verrà più modificato fino al termine della stessa. Dopo queste fasi il client è pronto a ricevere dati dal server. Per iniziare la ricezione di dati il client deve effettuare una richiesta di PLAY alla quale il server risponderà con l'invio di una risposta contenente informazioni che possono essere utilizzate dal client per gestire "riposizionamenti" nella riproduzione del contenuto multimediale. Da questo momento il client riceve dal server pacchetti di dati consegnati tramite RTP e suddivisi in base al tipo di payload del pacchetto. Inoltre vengono scambiate informazioni riguardanti la qualità del servizio offerto tramite RTCP. Alla richiesta di TEARDOWN il server risponde con un messaggio di risposta con codice di stato e termina l'invio dei dati tramite RTP.
Nella figura viene inoltre riportato con frecce tratteggiate il possibile utilizzo del protocollo SIP per l'inizializzazione di una sessione. Utilizzando SIP che è in grado di recuperare la descrizione dei file SDP gestiti dal server non è più necessario utilizzare il comando RTSP di DESCRIBE.

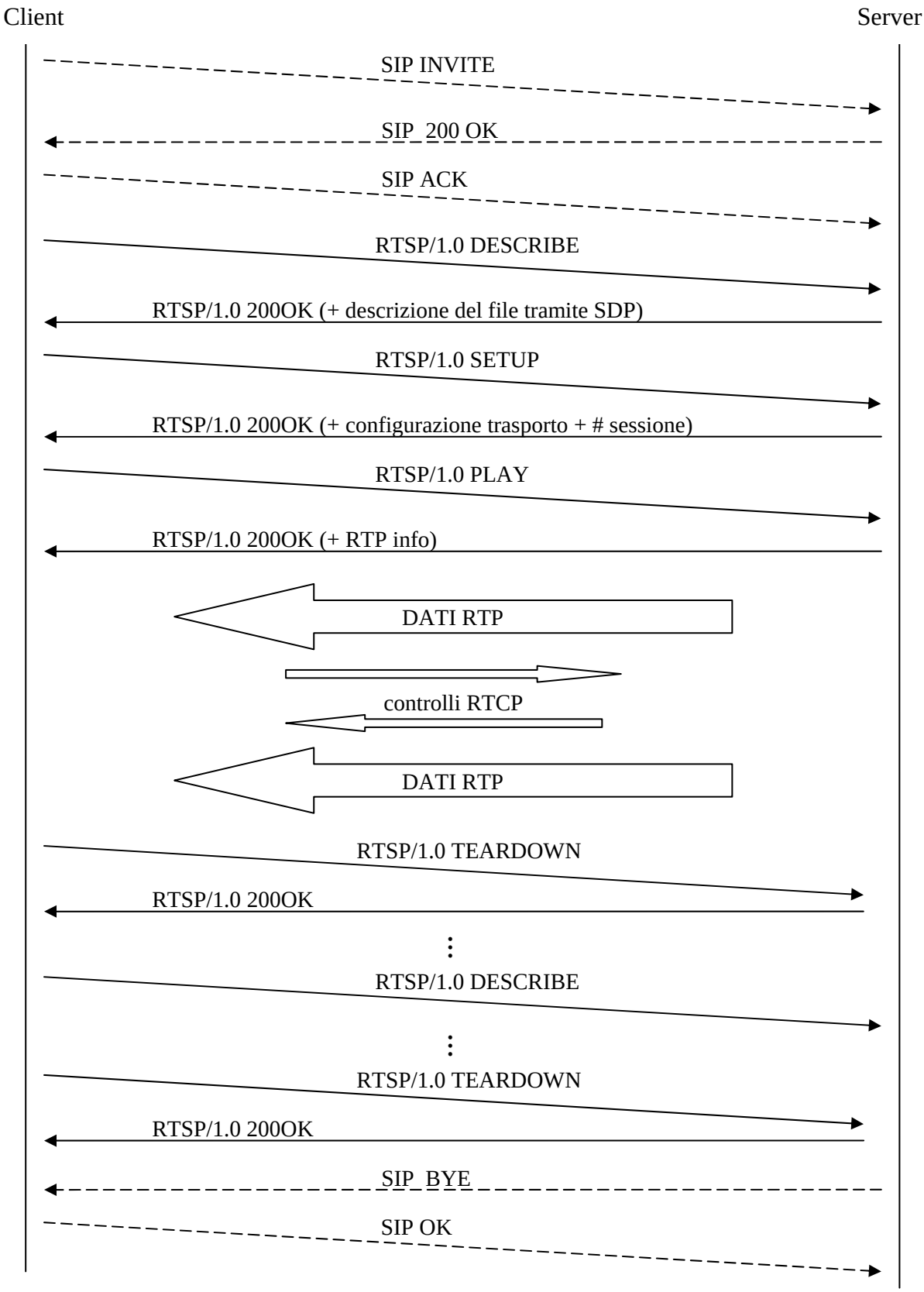

Figura 2.5 - Interazione tra i vari protocolli

3 Real Time Streaming Protocol

3.1 Analisi

Analizziamo ora più approfonditamente il protocollo RTSP già accennato nel capitolo precedente puntando la nostra attenzione in particolar modo sul formato e le funzioni dei vari messaggi che client e server si scambiano per l'inizializzazione, la gestione e la terminazione di una sessione multimediale.
Infine si vedranno alcune tecniche aggiuntive per la sicurezza delle sessioni RTSP.

3.2 Messaggio RTSP

I messaggi che vengono scambiati tra Client e Server sono basati su un protocollo testuale utilizzando il set di caratteri ISO 10646 codificati in UTF-8. I messaggi possono essere suddivisi in due gruppi: Messaggi di richiesta (Message Request) e Messaggi di Risposta (Message Response) con una sintassi differente a seconda del caso. Generalmente il Server invia al client messaggi di risposta a seguito di una richiesta e deve essere in grado quindi di interpretare il contenuto dei messaggi ricevuti e inviare la risposta a seconda del caso. In certi casi il server può inviare al client messaggi di richiesta come nel caso, ad esempio, della redirezione su un altro server per uno o più file multimediali.

3.2.1 Messaggi di richiesta (Message request)

Il formato dei messaggi di richiesta prevede da parte del server il riconoscimento del metodo invocato, della risorsa specificata e la comprensione degli header generali, di richiesta e di entità.
Un messaggio di richiesta dal server al client e vice versa include nella prima linea di messaggio il metodo che deve essere applicato alla determinata risorsa espressa nell'URI e la versione del protocollo in uso. L'identificatore di fine linea è CRLF ma anche CR e LF devono essere riconosciuti come tali.
Il formato di un messaggio di richiesta è il seguente:

Request =
Request-Line
*(general-header | request-header | entity-header)
CRLF
[message-body]

3.2.2 Request line

La prima parte del messaggio di richiesta è la Request line che identifica il metodo invocato, la risorsa cui si riferisce e la versione del protocollo. In alcuni casi non è necessario che la richiesta si riferisca ad una risorsa specifica ma al server stesso. In questo caso infatti l'URI potrà essere costituita dall'indirizzo del solo server.

Request-Line = Method SP Request-URI SP RTSP-Version CRLF

METODI	
DESCRIBE	ANNOUNCE
OPTIONS	GET_PARAMETER
PAUSE	PLAY
PING	REDIRECT
SETUP	SET_PARAMETER
TEARDOWN	extension-method

Tabella 3.1 - Metodi

Request-URI = "*" | absolute_URI

RTSP-Version = "RTSP" "/" 1*DIGIT "." 1*DIGIT

3.2.3 Request Header Fields

REQUEST-HEADER	
Accept	Accept-Encoding
Accept-Language	Authorization
From	If-Modified-Since
Range	Referer
User-Agent	

Tabella 3.2 - Request Header Fields

Si noti che in contrapposizione con HTTP/1.1, RTSP richiede sempre l'URL assoluto (che è, incluso lo schema, host e porta), invece del percorso assoluto.
L'asterisco "*" nel Request-URI significa che la richiesta non deve essere applicata ad una particolare risorsa, ma al server stesso, ed è consentito solo quando il metodo non deve essere necessariamente applicato alla risorsa.
Un esempio può essere :

OPTIONS * RTSP/1.0

3.2.4 Messaggi di risposta (Message response)

I messaggi di risposta hanno una sintassi differente dei messaggi di richiesta e vengono utilizzati per rispondere alle richieste.
La sintassi di un messaggio di risposta è la seguente:
Response =
Status-Line
*(general-header | response-header | entity-header)
CRLF
[message-body]

3.2.5 Status Line

La status line è costituita dalla versione del protocollo e da un codice e relativo messaggio. Il codice identifica uno stato di errore o di successo del metodo richiesto mentre il messaggio testuale non ha una rilevanza nell'ambito del protocollo stesso. Non sono permessi CR o LF eccetto per la sequenza finale (CRLF).

Status-Line = RTSP-Version SP Status-Code SP Reason-Phrase CRLF

3.2.6 Codici e messaggi di stato (Status Code)

Il codice di stato è un elemento di tre cifre intere ed è il risultato di un tentativo di capire e soddisfare una richiesta. Il messaggio serve per dare una breve descrizione testuale del codice di stato.

La prima cifra del codice di stato indica la classe della risposta, ce ne sono 5 differenti:

- 1xx: Informational- Richiesta ricevuta il processo continua
- 2xx: Success- L'azione è stata ricevuta con successo, compresa e accettata
- 3xx: Redirection- Ulteriori azioni devono essere intraprese per completare la richiesta
- 4xx: Client Error- La richiesta contiene sintassi errata o non può essere soddisfatta
- 5xx: Server Error- Il server ha fallito nell'eseguire una richiesta apparentemente valida

I messaggi di stato sono solamente “raccomandati” e possono essere sostituiti con equivalenti senza avere effetti sul protocollo.
Si noti che RTSP adotta molti codici di HTTP/1.1 e RTSP aggiunge nuovi codici solo da x50 in poi per evitare conflitti con nuove definizioni di codici HTTP.
Nella successiva tabella vengono riportati i codici di errore tipici di RTSP:

STATUS-CODE	**DESCRIPTION**	
"100"	Continue	all
"200"	OK	all
"300"	Multiple Choices	all
"400"	Bad Request	all
"451"	Parameter Not Understood	SETUP
"452"	Conference Not Found	SETUP
"453"	Not Enough Bandwidth	SETUP
"454"	Session Not Found	all
"455"	Method Not Valid in This State	all
"456"	Header Field Not Valid for Resource	all
"457"	Invalid Range	PLAY
"458"	Parameter Is Read-Only	SET_PARAMETER
"459"	Aggregate operation not allowed	all
"460"	Only aggregate operation allowed	all
"461"	Unsupported transport	all
"462"	Destination unreachable	all
"500"	Internal Server Error	all
"551"	Option not supported	all

Tabella 3.3 – Codici e Messaggi di errore di RTSP (vengono riportati solamente i codici aggiunti da RTSP e quelli che definiscono le classi generali di errore)

I codici di stato di RTSP sono estensibili. Le applicazioni RTSP non devono comprendere il significato di ogni codice anche se tuttavia sarebbe preferibile. Comunque le applicazioni devono comprendere la classe del codice di stato che è indicata dalla prima cifra e devono trattare le risposte non riconoscibili come equivalenti al codice di stato x00 con x che rappresenta la classe del codice. Le risposte non riconoscibili non devono essere memorizzate.

3.2.7 Response Header Fields

Il campo response-header permette di inviare al destinatario informazioni addizionali che non possono essere contenute nella Status-Line. Questo campo da informazioni riguardanti il server e su un ulteriore accesso alla risorsa identificata dal Request-URI.

RESPONSE-HEADER	
Location	Proxy-Authenticate
Public	Retry-After
Server	Vary
WWW-Authenticate	

Tabella 3.4 - Response Header

3.3 Metodi

I metodi indicano quale azione occorre effettuare sulla risorsa identificata dall'URL di richiesta. I nomi dei metodi sono "case-sensitive". Nella successiva tabella vengono riportati tutti i metodi, nella quale vengono indicati quali sono quelli consigliati (recommended), opzionali, (optional) e quelli necessari (required).

METHOD	**DIRECTION**	**OBJECT**	**REQUIREMENT**
DESCRIBE	C->S	P,S	Recommended
ANNOUNCE	C->S, S->C	P,S	Optional
GET_PARAMETER	C->S, S->C	P,S	Optional
OPTIONS	C->S, S->C	P,S	Required (S->C: Optional)
PAUSE	C->S	P,S	Recommended
PLAY	C->S	P,S	Required
PING	C->S, S->C	P,S	Optional (C->S: Recommended)
REDIRECT	S->C	P,S	Optional
SETUP	C->S	S	Required
SET_PARAMETER	C->S, S->C	P,S	Optional
TEARDOWN	C->S	P,S	Required

Tabella 3.5 - Metodi (C: Client, S: Server, P: Presentazione, S: Stream)

3.3.1 OPTIONS

Rappresenta una richiesta di informazioni riguardo le opzioni della comunicazione disponibile nella catena richiesta-risposta identificata dalla URL specificata (non indicando una semplice URL è possibile anche usare una modalità per richiedere tutte le opzioni supportate dal server). Può essere emesso in ogni momento: per esempio se il client sta per provare una richiesta non standard. Non influenza lo stato del server.

Esempio:

C->S: OPTIONS * RTSP/1.0
CSeq: 1
Require: implicit-play
Proxy-Require: gzipped-messages

S->C: RTSP/1.0 200 OK
CSeq: 1
Public: DESCRIBE, SETUP, TEARDOWN, PLAY, PAUSE

3.3.2 DESCRIBE

Recupera da un server la descrizione di una presentazione o di un oggetto identificato da uno specifico URL. La coppia di messaggi che viene scambiata tra client e server costituisce la fase di inizializzazione di RTSP.

Esempio:

C->S: DESCRIBE rtsp://server.example.com/fizzle/foo RTSP/1.0
CSeq: 312
Accept: application/sdp, application/rtsl, application/mheg

S->C: RTSP/1.0 200 OK
CSeq: 312
Date: 23 Jan 1997 15:35:06 GMT
Content-Type: application/sdp
Content-Length: 376

v=0
o=mhandley 2890844526 2890842807 IN IP4 126.16.64.4
s=SDP Seminar
i=A Seminar on the session description protocol
u=http://www.cs.ucl.ac.uk/staff/M.Handley/sdp.03.ps
e=mjh@isi.edu (Mark Handley)
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
a=recvonly
m=audio 3456 RTP/AVP 0
m=video 2232 RTP/AVP 31
m=whiteboard 32416 UDP WB
a=orient:portrait

La risposta a DESCRIBE deve contenere tutte le informazioni necessarie all'inizializzazione per la risorsa che descrive, ovvero deve contenere tutti i parametri utili a gestire il tipo di informazioni richiesto. Naturalmente questo non è il solo modo per scambiare questo tipo di dati: si potrebbe ricorrere ad un altro tipo di protocollo (HTTP, e-mail con attachment); oppure dialogare tramite linea di comando o standard input.

3.3.3 ANNOUNCE

Questo tipo di Metodo serve a due scopi: se mandato da client ANNOUNCE annuncia al server la descrizione di una presentazione o di un oggetto multimediale identificato dalla URL richiesta. Se è mandato dal server ANNOUNCE aggiorna la descrizione della sessione in tempo reale.

Esempio:

C->S: ANNOUNCE rtsp://server.example.com/fizzle/foo RTSP/1.0
CSeq: 312
Date: 23 Jan 1997 15:35:06 GMT
Session: 47112344
Content-Type: application/sdp
Content-Length: 332

v=0
o=mhandley 2890844526 2890845468 IN IP4 126.16.64.4
s=SDP Seminar
i=A Seminar on the session description protocol
u=http://www.cs.ucl.ac.uk/staff/M.Handley/sdp.03.ps
e=mjh@isi.edu (Mark Handley)
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
a=recvonly
m=audio 3456 RTP/AVP 0
m=video 2232 RTP/AVP 31

S->C: RTSP/1.0 200 OK
CSeq: 312

3.3.4 SETUP

La richiesta di SETUP per un'URI specifica il meccanismo di trasporto che deve essere usato per il flusso di dati multimediali. Un client può fare una tale richiesta anche durante una esecuzione per cambiare i parametri di trasporto e il server dovrebbe consentirlo. Dal momento che SETUP contiene tutte le informazioni di inizializzazione, firewalls e altri dispositivi della rete sono risparmiati dall'analizzare anche la risposta DESCRIBE del server. Inoltre il server in risposta alla richiesta di SETUP genera ed invia l'identificatore della sessione.

Esempio:
C->S: SETUP rtsp://example.com/foo/bar/baz.rm RTSP/1.0
CSeq: 302
Transport: RTP/AVP;unicast;client_port=4588-4589

S->C: RTSP/1.0 200 OK

CSeq: 302

Date: 23 Jan 1997 15:35:06 GMT

Session: 47112344

Transport: RTP/AVP;unicast;

client_port=4588-4589;server_port=6256-6257

3.3.5 PLAY

Comanda al server di cominciare a mandare i dati attraverso il meccanismo di trasporto specificato in SETUP. Un client non deve mai emettere una richiesta di PLAY se non è stato ricevuto l'acknowledgment con successo dei messaggi di SETUP inviati.

Le richieste di PLAY possono essere accodate (pipeline) e il server deve rispondere secondo l'ordine delle richieste.

Se il trasferimento di dati è già stato inviato una seconda richiesta di PLAY viene servita quando la precedente richiesta viene completata.

Con questo comando è possibile specificare quando far partire una certa sequenza (Absolute Time) o a quali parti di quest'ultima accedere. Per fare questo esistono varie convenzioni per la scansione temporale di questi tipi di oggetti.

Esempi:

C->S: PLAY rtsp://audio.example.com/twister.en RTSP/1.0

CSeq: 833

Session: 12345678

Range: smpte=0:10:20-;time=19970123T153600Z

S->C: RTSP/1.0 200 OK

CSeq: 833

Date: 23 Jan 1997 15:35:06 GMT

Range: smpte=0:10:22-;time=19970123T153600Z

C->S: PLAY rtsp://audio.example.com/meeting.en RTSP/1.0
CSeq: 835
Session: 12345678
Range: clock=19961108T142300Z-19961108T143520Z

S->C: RTSP/1.0 200 OK
CSeq: 835
Date: 23 Jan 1997 15:35:06 GMT

3.3.6 PAUSE

Questo metodo causa l'interruzione temporanea della consegna dei dati senza liberare le risorse utilizzate. Nel caso di un singolo flusso vengono bloccati l'esecuzione e la registrazione. Nel caso di più flussi il comando provoca l'interruzione di tutti i flussi. Qualora entro un certo tempo (timeout) non venisse inviato un comando di PLAY il server può decidere di abbattere la connessione per liberare risorse per altri scopi. Altrimenti il comando di PLAY farebbe riprendere la consegna dei dati esattamente dall'interruzione: ovvero dal momento specificato nella richiesta.
Una richiesta di PAUSE elimina l’eventuale coda di richieste di PLAY. Il punto di pausa deve essere comunque mantenuto perché una seguente richiesta di PLAY senza Range header deve ripartire dal punto di pausa precedente.
Esempio:
C->S: PAUSE rtsp://example.com/fizzle/foo RTSP/1.0
CSeq: 834
Session: 12345678

S->C: RTSP/1.0 200 OK
CSeq: 834
Date: 23 Jan 1997 15:35:06 GMT

3.3.7 TEARDOWN

Ferma la consegna dei dati multimediali per una data URI, liberando le risorse associatele. Affinché una nuova sessione possa essere eseguita è necessaria una nuova negoziazione dei parametri tramite richiesta di SETUP.

Esempio:

C->S: TEARDOWN rtsp://example.com/fizzle/foo RTSP/1.0
CSeq: 892
Session: 12345678

S->C: RTSP/1.0 200 OK
CSeq: 892

3.3.8 GET_PARAMETER

La richiesta GET_PARAMETER recupera il valore di un parametro di una presentazione o di un flusso specificato dalla URI. Il contenuto della risposta è lasciato all'implementazione del server. Può essere usato per testare la presenza del client o del server (“ping”) nel caso si omettesse il corpo del messaggio.

Esempio:

S->C: GET_PARAMETER rtsp://example.com/fizzle/foo RTSP/1.0
CSeq: 431
Content-Type: text/parameters
Session: 12345678
Content-Length: 15
packets_received
jitter

C->S: RTSP/1.0 200 OK
CSeq: 431
Content-Length: 46
Content-Type: text/parameters
packets_received: 10
jitter: 0.3838

3.3.9 SET_PARAMETER

Richiede di settare il valore di un parametro per una presentazione o per un flusso specificato nella URI. Ogni richiesta dovrebbe contenere un singolo parametro per permettere così una migliore fruibilità delle risposte in caso di fallimento. Se una richiesta contiene più parametri il server deve soddisfare la richiesta solo se tutti i parametri possono essere settati correttamente.
Un server deve permettere che un parametro sia regolato ripetutamente allo stesso valore, ma può respingere i cambiamenti di valore del parametro.
Si noti che i parametri di trasporto per un flusso multimediale devono essere settati solo tramite il comando SETUP.

Esempio:
C->S: SET_PARAMETER rtsp://example.com/fizzle/foo RTSP/1.0
CSeq: 421
Content-length: 20
Content-type: text/parameters
barparam: barstuff

S->C: RTSP/1.0 451 Invalid Parameter
CSeq: 421
Content-length: 10
Content-type: text/parameters
barparam

3.3.10 REDIRECT

Informa il client che deve connettersi ad un altro server. Contiene le informazioni della locazione dell'altro server ed eventualmente l'istante in cui il servizio non sarà più disponibile. Se il client vuole continuare a mandare o ricevere dati con questa URI deve emettere un comando di TEARDOWN per la sessione corrente e un nuovo SETUP per la nuova sessione al nuovo indirizzo.

Esempio:

S->C: REDIRECT rtsp://example.com/fizzle/foo RTSP/1.0

CSeq: 732

Location: rtsp://bigserver.com:8001

Range: clock=19960213T143205Z-

3.3.11 PING

Questo metodo è un meccanismo bi-direzionale per controllare l'esistenza del client o del server durante la sessione. Non ha nessun effetto sulla sessione se non permettere a chi lo richiede se il client o il server sia ancora attivo. Chi richiede questo metodo deve includere l'header contenente la sessione con l'identificatore della sessione che deve essere controllata.

Esempio:

C->S: PING * RTSP/1.0

CSeq: 123

Session:12345678

S->C: RTSP/1.0 200 OK

CSeq: 123

Session:12345678

3.4 Header Field

Gli header field costituisco uno strumento per comunicare informazioni aggiuntive sull'oggetto a cui l'header stesso è associato. In particolare esistono quattro tipi diversi di header e più precisamente abbiamo: general header,request header, response header ed entity header field.

Molti header field di RTSP sono simili a quelli utilizzati in HTTP/1.1 per questo riportiamo nella seguente tabella solo quelli più significativi di cui diamo anche una breve descrizione:

HEADER	TYPE	SUPPORT	METHOD
Accept	general request-header	optional	entity
Accept-Language	request-header	optional	all
Allow	response-header	optional	all
Bandwidth	request-header	optional	all
Blocksize	request-header	optional	all but OPTIONS, TEARDOWN
Cache-Control	general request-header	optional	SETUP
Conference	request-header	optional	SETUP
CSeq	general request-header	required	all
Expires	entity-header field	optional	DESCRIBE, ANNOUNCE
Range	response-header	optional	PLAY, PAUSE,
Require	request-header	required	all
RTP-Info	response-header	required	PLAY
Scale	request-header response-header	optional	PLAY
Session	request-header response-header	required	all but SETUP, OPTIONS
Transport	request-header response-header	required	SETUP

Tabella 3.6 - Header Field ("entity" si riferisce a tutti i metodi che ritornano un MESSAGE BODY)

3.4.1 Accept

Tale campo nelle Request-header può essere usato per specificare i tipi di contenuto di certi "descrittori di presentazioni". Può essere usato per esempio per indicare che la richiesta in questione è limitata ad un ristretto insieme di tipi desiderati.

3.4.2 Accept-Language

E' simile ad Accept ma restringe l'insieme di lingue che sono preferite come risposta ad una domanda. La lingua specificata si riferisce esclusivamente al “descrittore della presentazione” e ad ogni frase di spiegazione e non al contenuto multimediale.

3.4.3 Allow

Elenca i metodi supportati dalla risorsa identificata dalla URI richiesta. Lo scopo è quello di informare in maniera puntuale il richiedente dei metodi validi associati alla risorsa.
Deve essere presente nelle risposte del tipo: *405 Method not allowed*.

3.4.4 Bandwith

Descrive la banda stimata disponibile al client espressa come un numero intero positivo e misurato in bit per secondo. Tale banda può cambiare durante una sessione RTSP.

3.4.5 Blocksize

E' mandato dal client al “media server” per chiedere una particolare dimensione dei pacchetti (non include gli header dei pacchetti dei livelli più bassi come IP, UDP o RTP). Il server è libero di utilizzare dimensione dei blocchi più piccoli di quelli richiesti; può troncare un pacchetto al multiplo della dimensione più piccola consentita dal mezzo usato o riscriverlo con la dimensione specificata dal mezzo usato. La dimensione del blocco è misurato in byte.

3.4.6 Cache Control

E' usato per specificare direttive a cui devono obbedire tutti i meccanismi di caching che si incontrano nella catena di richiesta-risposta.
Tali direttive devono passare attraverso applicazioni dei gateway o dei proxy senza preoccuparsi del significato di tali applicazioni dal momento che tali direttive possono essere applicate a tutti i destinatari lungo la catena richiesta-risposta. Non è possibile specificare una direttiva per una particolare cache.

Cache-Control dovrebbe essere specificata solo in una richiesta o risposta di SETUP; non governa il caching delle risposte ma piuttosto quello del flusso delle informazioni specificate dal SETUP. Le risposte a richieste RTSP non si possono salvare eccetto le risposte a DESCRIBE.

In particolare c'è la possibilità di indicare se flusso di dati multimediale non debba essere memorizzato da nessuna parte (*"no-cache"*) o se possa essere salvato da qualunque dispositivo con cache (*"public"*) o possa essere memorizzato solo dal singolo utente a cui è destinato che non abbia memoria condivisa (*"private"*).

Con la specifica "*no-transform*" si obbligano i dispositivi intermedi a non convertire per nessun motivo il formato dei dati disponibili in un altro. La situazione tipica potrebbe essere un proxy che converta i formati video per salvare spazio in cache o ridurre la quantità di traffico su un link lento. Nascerebbero seri problemi qualora le trasformazioni fossero fatte su flussi dedicati a certi tip di applicazioni. Per esempio immagini per applicazioni mediche, analisi di dati scientifici, applicazioni che richiedono autenticazione tra utenti finali (end-to-end authentication).

In alcuni casi ad esempio qualora il collegamento alla rete risulti estremamente lento, il client può richiedere solo i dati multimediali che sono già stati salvati in memoria e non riceverli dal server originale. Per fare questo il client deve includere la direttiva "*only-if-cached*" nella richiesta. Se una cache riceve questa richiesta dovrebbe rispondere usando un flusso di dati multimediali che risponda alle altre limitazioni della richiesta o altrimenti rispondere con il codice di stato "*504 Gateway Timeout* ".

Con le direttive "*max-stale*" e "*min fresh*"il client può voler accettare flussi di dati multimediali che abbiamo oltrepassato il loro tempo di vita per un certo numero di secondi o flussi che non siano stati emessi dopo un certo numero di secondi dall'istante in cui si trova.

Con l'opzione "*must-revalidate*" una cache fa presente al client che ad ogni sua richiesta la cache per poter rispondere è costretta a riconoscere la validità di tale richiesta con il client.

3.4.7 Conference

Stabilisce una connessione logica tra una conferenza "pre-stabiltita" e uno stream RTSP.

L'identificatore della conferenza non deve cambiare nella stessa sessione RTSP.

Un codice di errore “452 Conference Not Found” viene inviato se l’identificatore della conferenza non è valido.

3.4.8 C-Seq

Specifica il numero di sequenza di una coppia di richiesta-risposta RTSP.
Tale campo deve essere presente in tutti i messaggi inviati tra client e server. Ad ogni richiesta contenete un specifico numero di sequenza deve corrispondere una risposta con la stesso numero. Nelle ritrasmissioni il numero non è incrementato.

3.4.9 Expires

Da una data e un tempo oltre il quale la descrizione o il flusso di dati multimediale deve essere considerato “vecchio”. L'interpretazione di questa specifica dipende da METHOD a cui ci riferiamo. La presenza di questo campo non implica che la risorsa originale cambierà o cesserà di esistere al termine di tale tempo.
Per esempio: *Expires: Thu, 01 Dec 1994 16:00:00 GMT*
Con opportune date c'è la possibilità di marcare da parte del server di origine una risposta come già “vecchia” o come “mai vecchia”.
La presenza di un Expires Header-Field garantisce la possibilità di salvare in cache i flussi di dati multimediali: qualora non ci fosse tale campo, a meno di altre indicazioni, il flusso è da ritenersi come non memorizzabile in cache.

3.4.10 Range

Specifica un intervallo di tempo. Tale intervallo può essere specificato in numero di unità secondo le specifiche SMPTE, NPT, Absolute Time. L'Header può anche contenere un parametro di tempo in UTC che specifica l'istante in cui l'operazione deve essere resa efficace. I server che supportano il Range Header devono conoscere i formati degli intervalli NPT e dovrebbero conoscere il formato degli intervalli SMPTE.
Il Range Header di risposta indica quale intervallo di tempo è attualmente in riproduzione o in registrazione. Qualora il formato dei tempi non è comprensibile il ricevente dovrebbe rispondere con il codice di stato: "*501 Not Implemented*".

3.4.11 Require

E' usato dai client per chiedere al server quali opzioni può o non può supportare. Il server deve rispondere usando gli Header appropriati (*Unsupported Header*) per negare la conoscenza di quelli che non supporta.

Serve per rendere veloce l'interazione tra client e server e fa risparmiare qualche ciclo di tempo rispetto ad una normale negoziazione, rimuove ambiguità quando il client richiede caratteristiche che il server non comprende.

Esempio:

C->S: SETUP rtsp://server.com/foo/bar/baz.rm RTSP/1.0
CSeq: 302
Require: funky-feature
Funky-Parameter: funkystuff

S->C: RTSP/1.0 551 Option not supported
CSeq: 302
Unsupported: funky-feature

C->S: SETUP rtsp://server.com/foo/bar/baz.rm RTSP/1.0
CSeq: 303

S->C: RTSP/1.0 200 OK
CSeq: 303

3.4.12 RTP-Info

Questo campo viene utilizzato per settare parametri specifici di RTP nella risposta al metodo PLAY:

- *url*: indica l'URL dello stream al quale corrispondono i seguenti parametri RTP
- *seq*: indica il numero di sequenza del primo pacchetto dello stream. Questo permette ai client di sistemare i pacchetti durante un "riposizionamento". Il client usa questo valore per differenziare i pacchetti precedenti e successivi al riposizionamento.

- *rtptime*:indica l'RTP timestamp corrispondente al valore di tempo contenuto nel Range response header. Il client usa questo valore per calcolare la corrispondenza del tempo RTP rispetto al Normal Play Time (NPT).

3.4.13 Scale

Un valore di 1 associato a Scale indica una riproduzione o registrazione a velocità normale. Qualora non fosse 1, per avere il valore indica la ragione per cui moltiplicare la velocità di normale visione. Un valore negativo indica direzione inversa.
Esempio di riproduzione in direzione inversa a 3.5 volte la velocità normale:
Scale:-3.5

3.4.14 Session

Questo tipo di campo Header sia in richiesta che in risposta identifica una sessione RTSP cominciata dal server di dati multimediali con una risposta di SETUP e conclusa con un messaggio di TEARDOWN. L'identificatore di sessione è un numero scelto dal server in maniera casuale: non deve farlo se comunque ha altri mezzi per identificare la sessione . Una volta che il client riceve un identificatore di sessione deve restituirlo in ogni richiesta associata a quella sessione.
In questo Header il server può indicare al client anche per quanto può aspettare tra un comando RTSP e un altro prima di chiudere la sessione per mancanza di attività. Tale timeout è misurato in secondi e il minimo di default è 60 secondi. Messaggi di controllo per più di una URL di tipo RTSP possono essere mandati all'interno di una singola sessione RTSP: questo perché è possibile che più client usino la stessa sessione per controllare più flussi di dati che costituiscono una presentazione finché provengono dallo stesso server. Naturalmente però sessioni di più utenti per la stessa URL provenienti dallo stesso client devono usare identificatori di sessione differenti.

3.4.15 Timestamp

Descrive quando il client ha mandato una richiesta al server. Il valore associato è importante solo per il client e può usare qualunque tipo di scala temporale. Il server deve ritornare lo stesso valore e può (qualora avesse informazioni precise) aggiungere un numero in virgola mobile con cui indicare il numero di secondi che sono passati da quando ha ricevuto la richiesta.

Il valore di Timestamp è usato dal client per calcolare il “round-trip-time” in maniera tale da modificare opportunamente il valore di tempo dopo il quale ritrasmettere.

3.4.16 Transport

Indica quale protocollo di trasporto deve essere usato e configura i suoi parametri come indirizzo di destinazione, compressione, mulitcast time-to-live, porta di destinazione per un singolo flusso di dati. Determina i parametri non ancora fissati dalla descrizione di presentazione. I tipi di trasporto sono elencati in ordine di preferenza e per ognuno di essi possono essere indicati anche alcuni parametri.

Può essere usato per cambiare alcuni parametri di trasporto. Il server può rifiutare questa opzione indicando nella risposta i valori attualmente scelti.

Può contenere una lista di opzioni di trasporto accettabili dal client.

La sintassi per la specifica di trasporto è *Transport/profile/lower-transport.*

Il valore di default per il *lower-transport* dipende dal profilo (per es: RTP/AVP richiede UDP).

I parametri di configurazione associati al transporto prevedono indicazioni su :

- tipo di consegna dei dati: *multicast* o *unicast*.
- *indirizzo* a cui mandare il flusso di dati: per questioni di sicurezza un server dovrebbe sempre autenticare il client in questione prima di permettere al client di re-indirizzare il flusso di dati multimediali ad un indirizzo diverso da quello scelto dal server, e non dovrebbe mai permettere il re-indirizzamento su un indirizzo che non sia quello da cui provengono i comandi. Questo è particolarmente importante in caso i comandi siano emessi via UDP.
- i metodi che devono essere supportati in questa sessione.
- canali da usare nel caso di *Dati binari incapsulati* : questo permette di gestire RTP/RTCP in maniera simile a come è gestito con UDP: un canale per RTP e un altro per RTCP
- Specifiche su RTP: coppia di porte per una sessione multicast, coppia di porte per unicast RTP/RTCP su cui il client (o il server)ha deciso di ricevere i dati multimediali e le informazioni di controllo,valore del parametro RTP SSRC usato dal server in trasmissione unicast che identifica la sorgente di sincronizzazione associata al flusso di dati multimediali.

Transport Header è ristretto a descrivere un singolo flusso RTP. (RTSP può anche controllare flussi multipli come una singola entità). Facendolo parte di RTSP piuttosto che confidare su una moltitudine di formati di descrizione di sessione facilita di molto il disegno dei *firewall*.

3.5 Considerazioni sulla sicurezza

RTSP riutilizza molte delle caratteristiche di sicurezza di HTTP quali:

- Meccanismi di autenticazione
- Privacy dei dati trasferiti
- DNS spooning

Inoltre ne presenta di nuove:

- Concentraded denial-of-service (DoS) attack: Il protocollo è soggetto agli attacchi DoS se un nodo “infetto” inizia un traffico di dati da uno a molti indirizzi IP posti come destinazione in una richiesta di SETUP. Un nodo può essere raggiunto da uno stream non richiesto senza rendersene conto per l’intervento del codice “infetto”. Una soluzione per questo problema sta nell’invio da parte del server di dati solo dopo aver identificato il client.
- Session hijacking: sessione e livello di trasporto sono non correlati e quindi è possibile da un nodo “in ascolto” redirigere verso s’è stessi la sessione RTSP corrente inviando una richiesta al server con lo stesso numero di sequenza della sessione “hijacked”. Per tentare di ridurre questa possibilità il server può usare un numero casuale di sequenza difficilmente identificabile.
- Persistently suspictius behaviour: Una delle caratteristiche di sicurezza del protocollo è la possibilità dei server di rifiutare le richiesta dei client se queste possono comportare rischi per la sicurezza. In casi “estremi” i server possono far abortire la sessione se ricevono richieste sospette e possono rispondere al client con un codice di errore “403 forbidden”. I server possono inoltre chidere la connessione con il client e proibire ogni futuro tentativo di connessione.

4 Java Media Framework

4.1 Introduzione

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.
Le API JMF offrono al programmatore un potente mezzo per sviluppare applicazioni e applet in grado di catturare, riprodurre, trasmettere e transcodificare contenuti multimediali.
Il trattamento di contenuti multimediali richiede un'elevata velocità computazionale (e.g. decompressione delle immagini, rendering) quindi non è sufficiente la sola emulazione della CPU come avviene in Java per ottenere prestazioni ottimali. Quindi un programmatore che desideri implementare un lettore multimediale in Java e voglia ottenere eccellenti prestazioni, deve necessariamente ricorrere a codice nativo della piattaforma alla quale è interessato. Questo comportamento ha come conseguenza che un programma Java con codice nativo non è più portabile su piattaforme differenti oltre a richiedere una buona conoscenza delle funzioni native da parte del programmatore.
Le API JMF tentano di risolvere questo problema, mettendo a disposizione del programmatore una serie di chiamate ad alto livello per la gestione del codice nativo.

4.2 Modello di elaborazione dei dati

Il JMF si basa su un modello di elaborazione dei dati (figura 4.1) in cui esistono tre differenti sezioni:

- la prima sessione è quella di *input* dove possiamo avere un file presente su disco oppure acquisito tramite dispositivo esterno o tramite la rete
- una fase intermedia o di *process* dove si può manipolare il flusso di dati
- una fase finale di *output* dove il flusso multimediale può essere presentato, salvato su file o spedito attraverso la rete.

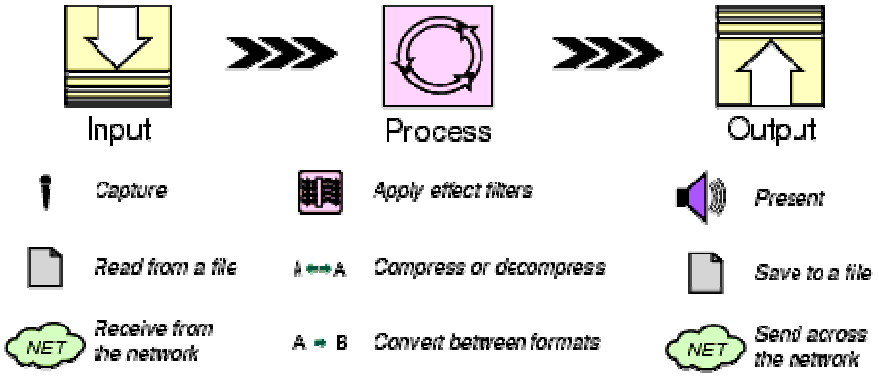

Figura 4.1- Modello di elaborazione dei dati

Il JMF fornisce un'architettura unificata per la gestione di cattura, elaborazione e consegna di dati multimediali basati sul tempo. JMF può supportare molti formati standard, come AIFF, AU, AVI, GSM, MIDI, MPEG, QuickTime, RMF e WAV.

4.3 Architettura JMF

Dispositivi come i videoregistratori forniscono un modello familiare per l'elaborazione, la registrazione e la distribuzione di dati multimediali. Quando si visualizza un filmato usando un videoregistratore il flusso di dati è fornito al dispositivo mediante l'inserimento di una video cassetta. Il videoregistratore legge e interpreta i dati dalla cassetta e li traduce in segnali appropriati che trasmette attraverso la televisione.

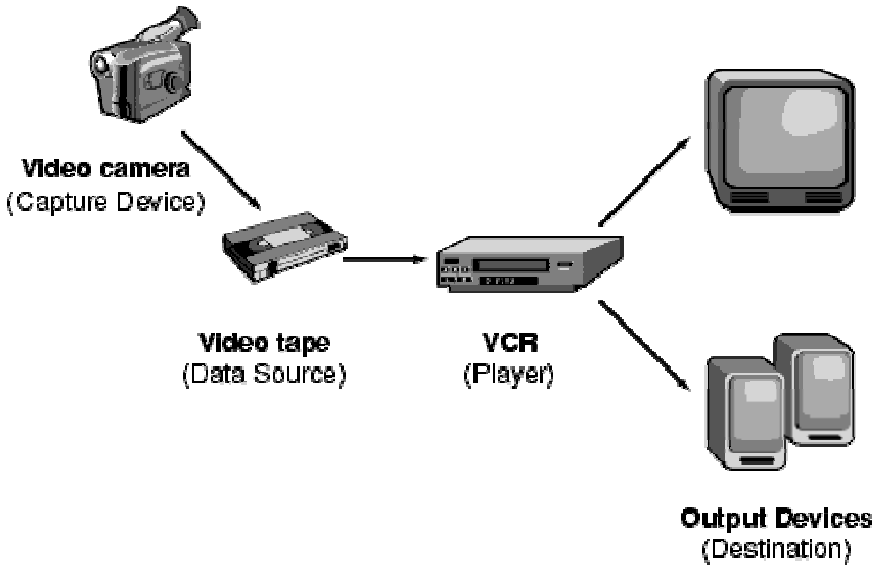

Figura 4.2 - Modello di base del JMF per la gestione dei dati multimediali

Il JMF utilizza lo stesso modello di base che viene rappresentato in figura 4.2. Dal modello si può comprendere come vengono trattati i vari dispositivi:

- ogni periferica di acquisizione è vista come un oggetto "*Capture Device*" che fornisce dati multimediali da elaborare
- l'astrazione di ogni sorgente dati viene vista come un oggetto di tipo "*DataSource*"
- il visualizzatore dei flussi audio e video è un oggetto chiamato "*Player*" che elabora il "*DataSource*" decodificando il flusso di dati multimediali e trasmettendoli ai dispositivi di output

DataSource e *Player* fanno parte delle API di alto livello del JMF . Il JMF fornisce inoltre delle API di basso livello che supportano l'integrazione nei programmi Java di componenti di elaborazione ed estensioni personalizzate. Questa stratificazione permette ai programmatori Java di avere a disposizione della API semplici da utilizzare e allo stesso tempo mantiene la flessibilità e l'estendibilità necessari per supportare applicazioni avanzate e tecnologie future.

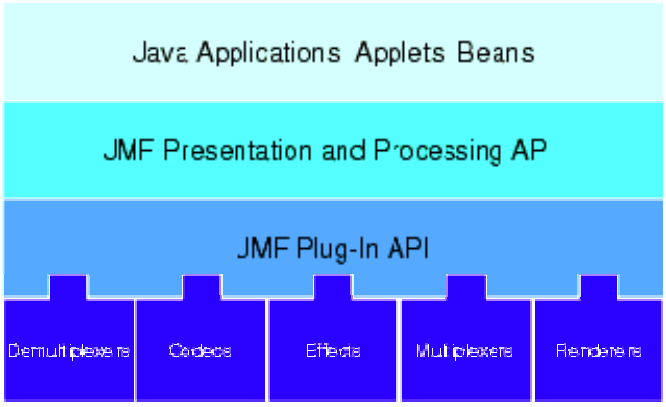

Figura 4.3 - Architettura JMF di alto livello

Le API del JMF consistono principalmente di interfacce che definiscono il comportamento e l'interazione di oggetti usati per gestire dati multimediali. L'implementazione di queste interfacce avviene all'interno della struttura del framework. Usando oggetti intermediari chiamati m*anager*, JMF rende più semplice l'integrazione di nuove implementazioni di interfacce chiave che possano essere usate con le classi esistenti.
JMF utilizza quattro gestori (*manager)*:

- **Manager**: gestisce la creazione di *Player*, *Processor*,*DataSource* e *DataSink*.
- **PackageManager**: mantiene un registro che contengono le classi del JMF

- **CaptureDeviceManager**: mantiene un registro dei dispositivi di acquisizione disponibili
- **PlugInManager**: mantiene un registro dei componenti plug-in di elaborazione del JMF, come *Multiplexer, Demultiplexer, Codec, Effect* e *Renderer.*

Per creare oggetti di tipo Player, Processor, DataSource e DataSink nei programmi basati su JMF, è necessario richiamare il metodo create dell'oggetto Manager. Il CaptureDeviceManager dà invece informazioni relative agli apparati di cattura disponibili nel caso si vogliano ricevere in ingresso dati audio e video.

4.3.1 Modello del tempo

JMF misura il tempo con la precisione del nanosecondo. Un punto particolare nel tempo viene rappresentato con un oggetto di tipo Time. Le classi che supportano il modello del tempo di JMF (figura 4.4) implementano l'interfaccia Clock per rappresentare lo scorrere del tempo di un particolare flusso di dati. L'interfaccia Clock definisce la base del tempo e i meccanismi di sincronizzazione necessari a controllare la riproduzione di un file multimediale.

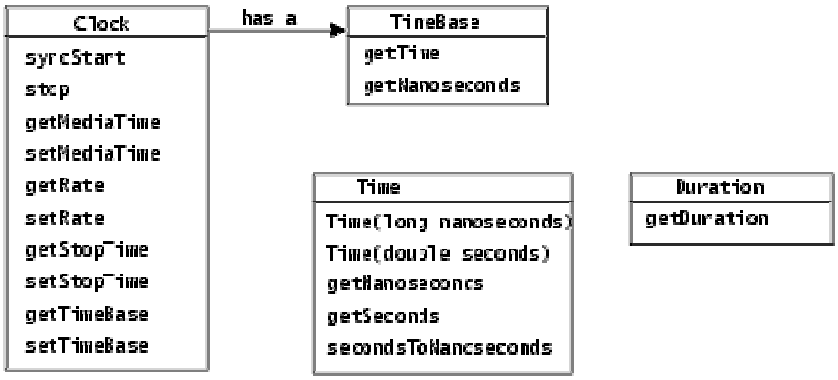

Figura 4.4 – Interfacce e classi JMF per la gestione del tempo

Il sistema mantiene un'interfaccia master, *TimeBase,* che misura il tempo, con l'accuratezza del nanosecondo, a partire dal 1° gennaio del 1970; questo tempo, chiamato time-base time, non può essere bloccato, né resettato e si basa sul clock di sistema.

L'oggetto *MediaTime* è invece una rappresentazione temporale della vita del Player e cioè del flusso dei dati multimediali che vengono riprodotti. Fissare un *MediaTime* per il Player è come settare una porzione di lettura all'interno dello stream dati, visto come sequenza di informazioni che necessitano di un certo tempo per essere eseguite. Il *MediaTime* associato al Player è infatti limitato ed il limite è dato dalla fine del flusso di dati stesso. Il *MediaTime* è come una sovrapposizione della vita temporale del media su il *TimeBase* che rappresenta lo scorrere del tempo.
L'interfaccia Clock definisce una mappatura tra *TimeBase* e *MediaTime*.

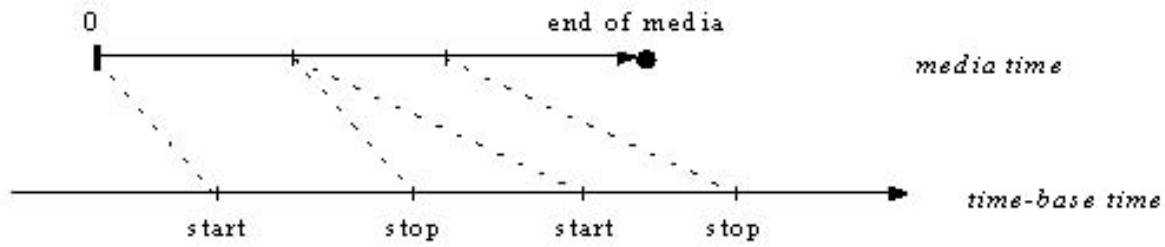

Figura 4.5 - Modello del tempo JMF

4.3.2 Modello degli eventi

JMF utilizza un meccanismo di segnalazione di eventi per mantenere i programmi informati sullo stato corrente del sistema e per permettere ai programmi stessi di rispondere alle condizioni di errore (mancanza di dati o risorse non più disponibili) che si verificano nella gestione dei dati multimediali.
Ogni volta che un oggetto del JMF deve fornire informazioni sulla condizione corrente del sistema 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. L'oggetto che spedisce eventi è un controller e gli oggetti che vogliono ricevere l'evento devono estendere *ControllerListener* e registrarsi presso il *Controller* (figura 4.6)

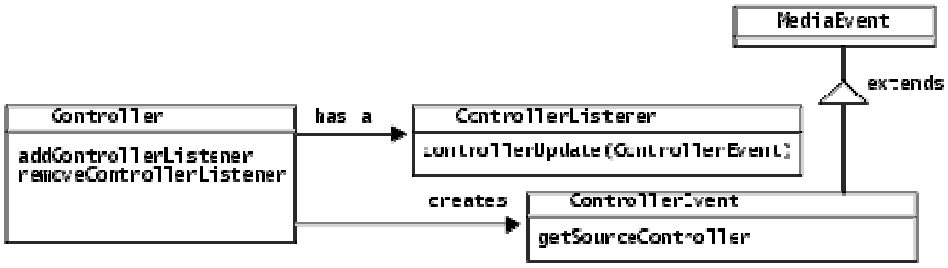

Figura 4.6 - Schema del modello ad eventi del JMF

4.3.3 Modello dei dati

I Player JMF generalmente usano un oggetto *DataSource* per gestire il trasferimento dei dati multimediali. Un *DataSource* è l'astrazione utilizzata la framework per incapsulare sia la posizione sia il protocollo e il software utilizzati per la consegna dei dati; può essere identificato mediante un oggetto *MediaLocator* (descrittore del contenuto di un media) o da un URL (universal resource locator) dal quale si può comunque costruire un oggetto *MediaLocator*.

Un *DataSource* gestisce internamente un elenco di oggetti di tipo *SourceStream* che possono utilizzare un elenco di Byte come unità di trasferimento o un Buffer appropriato a seconda del tipo di *DataSource* (figura 4.7).

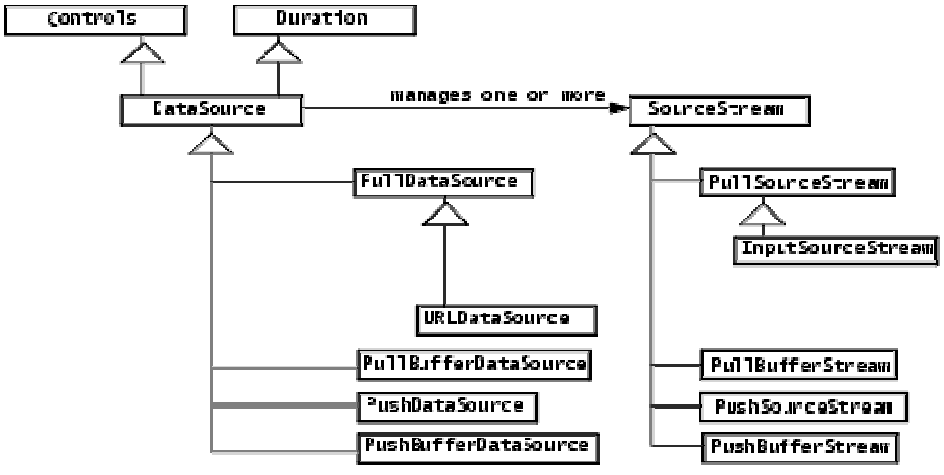

Figura 4.7 - Modello dei dati JMF

Gli stream di dati possono essere distinti a seconda delle caratteristiche di trasmissione:

- *pull*, se la trasmissione è avviata e gestita dal client
- *push*, se invece la trasmissione viene iniziata e controllata dal server

4.3.4 Formato dei dati

Il JMF per rappresentare il formato dei dati 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.

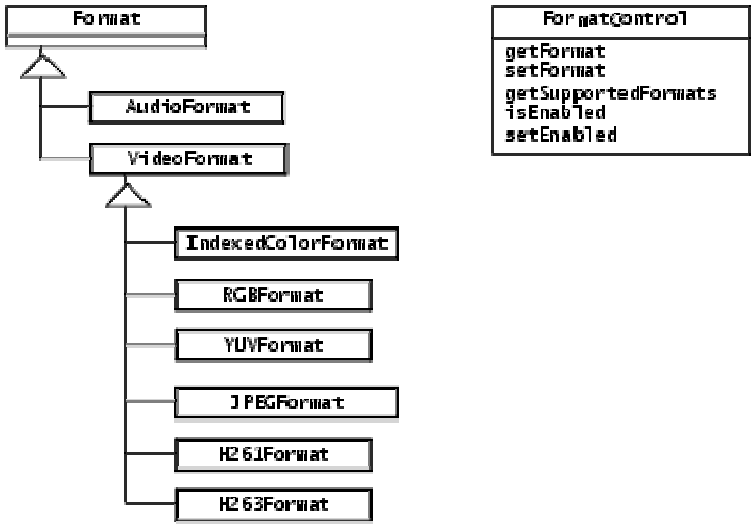

Figura 4.8 - Formato dei dati JMF

4.3.5 Controlli del JMF

JMF *Control* fornisce un meccanismo per l'impostazione e l'interrogazione degli attributi di un controllo. Un controllo fornisce spesso l'accesso al componente di tipo interfaccia utente che permette il controllo degli attributi di un oggetto.
Il JMF contiene le interfacce per i controlli mostrati in figura:

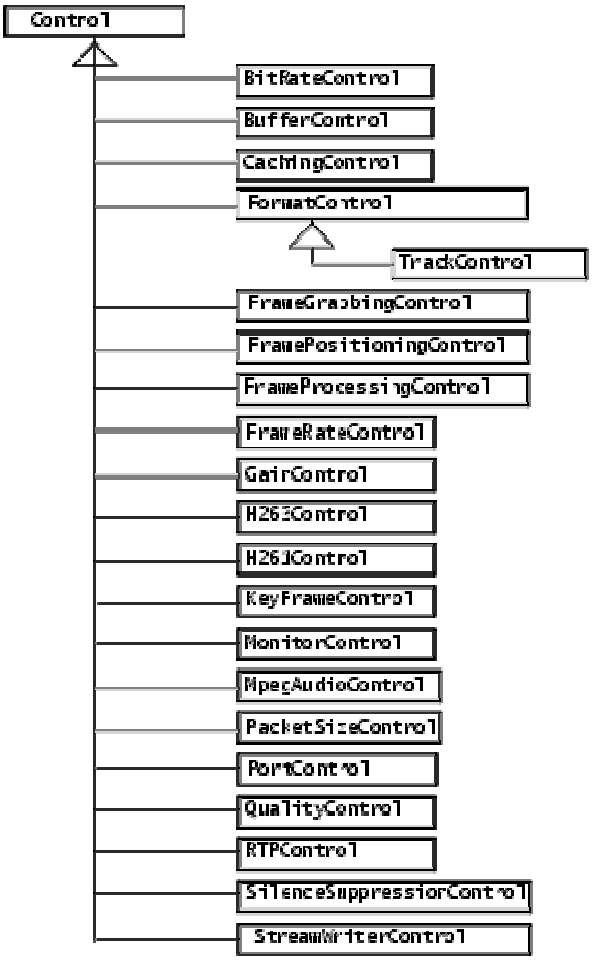

Figura 4.9 - Controlli standard del JMF

4.4 I componenti di interfaccia di utente

L'oggetto *Control* di JMF fornisce un meccanismo per settare e richiedere particolari attributi di un oggetto. Questo oggetto permette inoltre di accedere ad un componente di interfaccia di utente, *Component,* il quale fornisce all'utente delle funzioni di controllo sugli attributi dell'oggetto corrispondente. Il componente di default per un particolare *Control*, si ottiene invocando il metodo *getControlComponent* di quel *Control*.

Il metodo restituisce un *Component AWT* che può essere aggiunto alla propria finestra di applicazione. L'oggetto *Player,* ad esempio, fornisce l'accesso sia ad un componente di visualizzazione, sia ad un componente di controllo; per ottenere questi componenti vanno invocati i metodi getVisualComponent e getControlPanelComponent del Player.

Se non si vogliono usare i componenti di controllo di dafault forniti da una particolare implementazione, se ne possono implementare di personalizzati.

4.5 Presentazione dei dati multimediali

In JMF il processo di presentazione dei dati è modellato dall'interfaccia *Controller*.
Le API del JMF definiscono due tipi di controller: *Player* e *Processor* (figura 4.10). Entrambi sono generalmente realizzati per specifici *DataSource* e normalmente non vengo riutilizzati per presentare altri dati multimediali

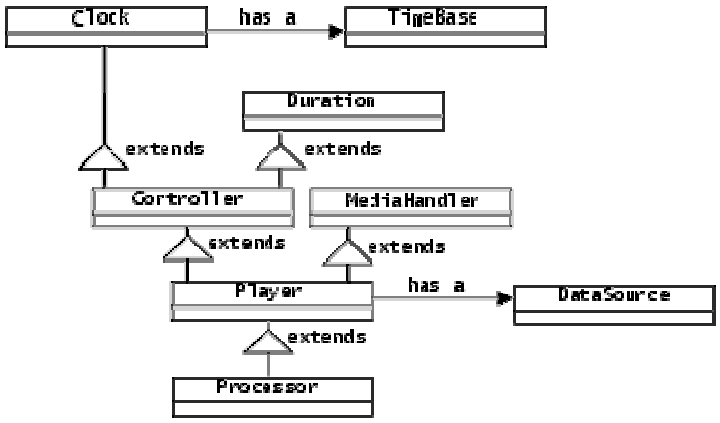

Figura 4.10 -Controller del JMF

4.5.1 Player

Il Player è un API che elabora i flussi di dati in ingresso e li riproduce in un determinato momento. Il Player riceve in ingresso un oggetto *DataSource* e il dispositivo di visualizzazione dipende dal tipo di dato da riprodurre come si può vedere nella figura successiva.

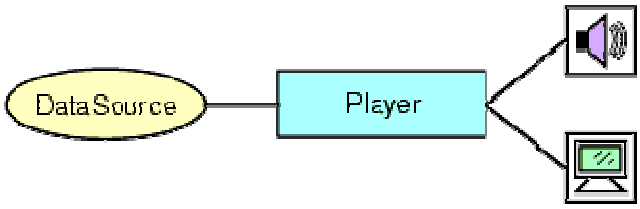

Figura 4.11 - Modello del Player

L'esistenza del Player viene suddivisa in sei differenti stati (figura 4.12), ciascuno dei quali è caratterizzato da diverse operazioni che è possibile effettuare.

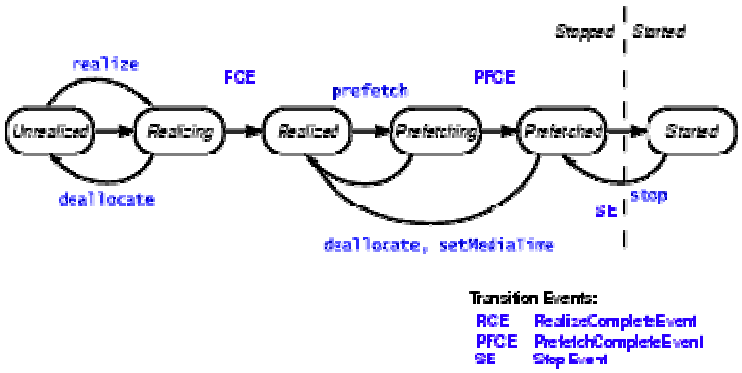

Figura 4.12 - Stati del Player

Gli stati sono:

- *Unrealized:* il Player si trova in questo stato all'atto della sua creazione, cioè quando è solamente a conoscenza dell'URL della risorsa da riprodurre. Un opportuno Manager è responsabile della creazione del Player attraverso il metodo createPlayer(), il cui unico parametro è proprio l'URL associato
- *Realizing:* una volta che il Player è stato creato, attraverso il metodo realize() dell'interfaccia Controller, è possibile portarlo in questo stato. Qui il Player acquisisce le risorse di sistema che utilizzerà durante la riproduzione, ma che non sono utili ad altri eventuali riproduttori. Si dice cioè che il Player si impossessa delle risorse non esclusive
- *Realized:* una volta che il Player ha acquisito le risorse relative alla fase precedente, si porta automaticamente nello stato *Realized.* Qui il Player è a conoscenza del suo compito, perciò è qui che si rendono disponibili i componenti *ControlPanelComponent* e *VisualComponent.* Un Player nello stato *Relized* non blocca nessun altro Player candidato alla riproduzione del media, per cui più Player contemporaneamente possono esistere in questo stato
- *Prefetching:* quando il Player è nello stato *Realized*, è possibile portarlo nello stato di *Prefetching* utilizzando il metodo prefetch(). Questo stato è caratterizzato dal fatto che il Player comincia ad acquisire le risorse per la riproduzione del file, comprese le risorse esclusive, se disponibili. Esso si riserva anche un certo buffer in cui farà il precaricamento dei primi dati da riprodurre

- *Prefetched:* terminata la fase precedente, si ha il passaggio automatico nello stato *Prefetched.* Qui il Player è pronto a partire. Tutte le risorse necessarie sono state acquisite ed eventuali altri Player che avessero bisogno delle stesse risorse sarebbero impossibilitati all'esecuzione
- *Started:* finalmente, attraverso il metodo start(), è possibile iniziare la riproduzione della risorsa attraverso il Player

Il passaggio tra gli stati è notificato attraverso l'emissione di un messaggio di evento di tipo *TransitionEvent*. L'interfaccia *ControllerListener* permette dunque di determinare in ogni momento in quale stato si trovi il Player e di agire di conseguenza.

Per il passaggio da uno stato all'altro esistono cinque metodi principali che, per evitare situazioni di conflitto, non possono essere chiamati in uno stato qualunque, pena la generazione di un'eccezione:

- start(): appartiene all'interfaccia *Clock* e può essere eseguito solo quando il Player è nello stato *Prefetched*. Lo stato seguente è *Started*.
- stop(): appartiene all'interfaccia Clock e può essere eseguito solo quando il *Started*. Lo stato seguente è *Prefetched*.
- realize(): serve per portare il Player dallo stato *Unrealized* allo stato *Realizing.*
- prefetch(): serve per iniziare la fase di *prefetching.*
- deallocate(): serve per liberare le risorse esclusive acquisite dal Player.

4.5.2 Processor

Un Processor è una specializzazione della classe *Player*; pertanto può fare tutto ciò che permette un Player, ma in aggiunta fornisce un flusso multimediale attraverso un *DataSource* che può essere presentato da un altro Player, Processor o spedito attraverso un *DataSink* (figura 4.13).

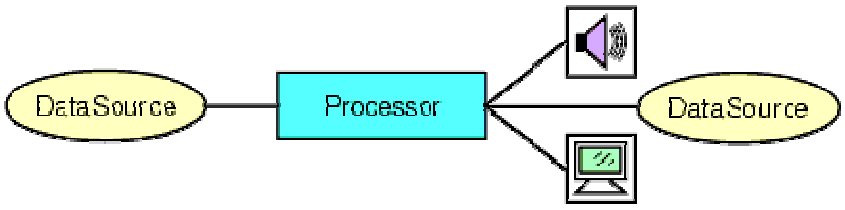

Figura 4.13 - Modello del Processor JMF

Mentre l'elaborazione del Player è predefinita dall'implementazione, il Processor consente allo sviluppatore dell'applicazione di personalizzare il tipo di trattamento da applicare ai dati multimediali in ingresso (figura 4.14).

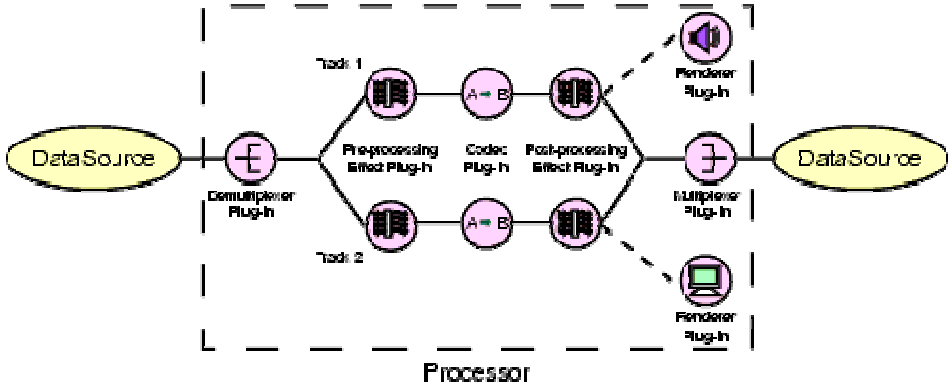

Figura 4.14 - Schema dettaglaito del Processor

Anche il Processor viene rappresentato in stati e rispetto al Player ne sono stati aggiunti due che vengono evidenziati in rosso nella figura seguente:

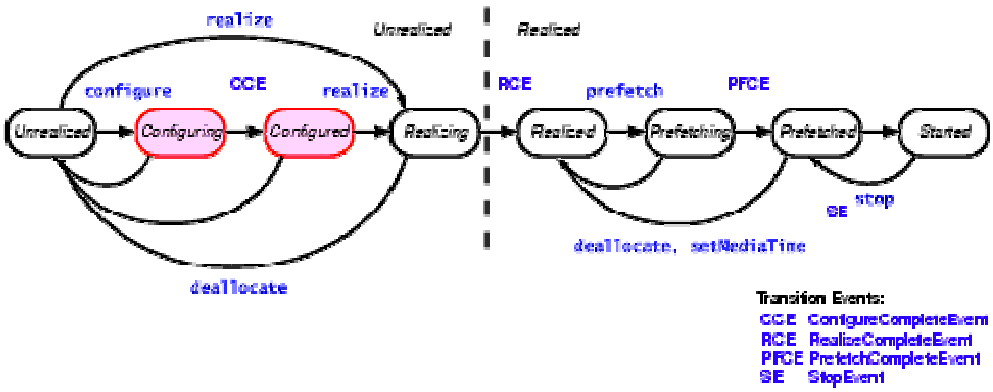

Figura 4.15 - Stati del Processor

I due stati aggiunti sono:

- *Configuring*: in questo stato avviene il collegamento con il *DataSource* e il "de-multiplexaggio" degli stream di input e si ottengono le informazioni riguardanti il formato dell'input
- *Configured*: il Processor si porta in questo stato a seguito del collegamento con il *DataSource* e dopo aver determinano il formato dei dati; qui viene emesso un evento *ConfigureCompleteEvent*

I passaggi tra i vari stati avvengono tramite i seguenti metodi che si aggiungo a quelli già visti nel Player:

- configure(): permette il passaggio dallo stato *Unrealized* allo stato *Configuring*
- realize(): serve per portare Processor dallo stato *Configured* allo stato *Realized*

4.6 Supporto JMF per RTSP

Le api di Java Media Framework offrono la possibilità di creare client RTSP sia per applicazioni che per applet che comunichino con un server RTSP da cui poter richiedere lo streaming di un particolare contenuto multimediale.

Per costruire un player RTSP, che riceva dati e da un server, si deve utilizzare il metodo *Manager.createPlayer* e passare come argomento l'URL del server da cui vogliamo ricevere lo streaming.

Attraverso questo metodo la JMF richiama metodi di una serie di classi che si occupano dell'inizializzazione, del controllo e della ricezione di dati.

Le principali classi coinvolte sono:

- *Manager* contenuta nel package javax.media. Questa classe definisce dei metodi per la creazione del player. Questi metodi utilizzando la reflection istanziano un oggetto DataSource e creano il player in base al tipo di DataSource appena istanziato.
- *DataSource* contenuta nel package com.sun.media.protocol.rtsp. Definisce un *DataSource* specifico per il protocollo RTSP.
- *Handler* contenuta nel package com.sun.media.content.rtsp.handler. Questa classe si occupa della gestione dell'intera sessione RTSP. In particolare richiede la creazione della connessione al server tramite un metodo della classe *Connection*, se la connessione è avvenuta con successo richiama un metodo della classe *RtspUtil* richiedendo l'inizializzazione del protocollo RTSP attraverso i metodi del protocollo DECRIBE e SETUP. Se anche questa operazione non genera errori vengono salvati il numero di tracce audio video da riprodurre e le porte del server dalle quali ricevere gli stream di dati e il controllo per i dati stessi. Vengono inoltre settati i buffer e le soglie minime per l'audio e il video. Gestisce inoltre le operazioni di PLAY PAUSE e TEARDOWN richiamando sempre metodi della classe *RtspUtil*

- *RtspUtil* contenuta nel package com.sun.media.content.rtsp. Questa classe fornisce metodi per la creazione e l'invio dei messaggi tipici del protocollo RTSP.
- *RtspManager* contenuta nel package com.sun.media.rtsp Questa classe gestisce le connessioni e invia i messaggi.
- *Connection* contenuta nel package com.sun.media.rtsp. Si occupa della creazione della connessione vera e propria al server RTSP attraverso l'utilizzo delle Socket.

Il JMF inoltre mette a disposizione all'interno del package com.sun.media.rtsp.protocol le classi per creare i metodi, gli header e i messaggi di richiesta e risposta.

5 Tecnologie e Strumenti

5.1 Introduzione

Lo scopo principale di questa tesi è quello di capire attraverso quali classi la Java Media Framework gestisce e utilizza il protocollo RTSP lato client. Inoltre per dare dimostrazione delle capacità di tali classi si realizzerà una semplice applicazione in grado di comunicare con un server che supporti i protocolli RTSP RTP e RTCP.
Per poter realizzare questi obbiettivi è necessario avere un idea delle tecnologie e degli strumenti utilizzati ed in questo capitolo ne vengono evidenziati gli aspetti di maggior rilevanza.

5.2 Formati audio video e codec

5.2.1 Formati audio

I formati audio principali sono il Wave e l'AIFF, rispettivamente per piattaforma Windows e Macintosh, le cui estensioni sono .wav e .aif. Entrambi derivano da un formato più generico, l'IFF (*Interchange File Format*), che salva i dati in blocchi di memoria detti *chunk*.
Ogni chunk è costituito da una serie di byte e serve ad uno scopo diverso: all'inizio del file, ad esempio, i chunk contengono informazioni identificative che specificano, tra le altre cose, il tipo di file, poi vengono i chunk contenenti i dati veri e propri. I formati Wave e AIFF distribuiscono i dati in questo modo. Le loro caratteristiche sono:

- versatilità
- capacità di supportare suoni mono e stereo
- possibilità di supportare diverse risoluzioni e frequenze di campionamento

Frequenza (kHz)	Risoluzione (bit)	Qualità
48	16	DAT (Digital audio tape)
44,1	16	CD audio
22,050	16	Musicassetta
11,025	16	Televisore
11,025	8	Conversazione telefonica

Tabella 5.1-Qualità dell'audio digitale in relazione alla frequenza e alla risoluzione

La Tabella 5.1 offre lo spunto per introdurre un altro argomento fondamentale quando parliamo di audio in relazione ad Internet, ovvero la dimensione dei file. Più sono alti i due parametri di cui si è parlato sopra, più crescono anche le dimensioni del file che si ottiene. Esiste una semplice formula, riportata di seguito, che permette di calcolare approssimativamente le dimensioni di un file audio.

dimensione del file = (durata in secondi) × (numero di canali) × (frequenza di campionamento) × (risoluzione) / 8

Ad esempio, un minuto di audio mono, qualità CD (60 secondi, 1 canale, 44,1 kHz, 16 bit) occuperebbe: 60 × 1 × 44,1 × 16 / 8 = 5,3 Mb

Nel caso di un file stereo, in cui il numero di canali raddoppia, si avrebbe un file di 10,6 Mb. Se si pensa che una canzone dura mediamente 3 minuti e mezzo (210 secondi), si avrebbero file di dimensioni pari a circa 37 Mb. Mentre queste dimensioni non rappresentano un problema in locale, diventano veramente eccessive quando si parla di Internet. Per questo si ragiona in termini di compressione e sono stati sviluppati dei formati *ad hoc* per Internet.

5.2.2 Formato MP3

Questo formato è stato sviluppato dal gruppo di ricerca Fraunhofer IIS, che fa parte dell'organizzazione MPEG(*Moving Picture Expert Group*) e si basa su un algoritmo di compressione chiamato *MPEG 1 layer III*, uno schema particolare di compressione dello standard più generale MPEG-1 applicato all'audio che si basa su criteri di tipo psicoacustico: questi considerano l'orecchio umano incapace di percepire sia le frequenze superiori alla cosiddetta soglia di udibilità che le frequenze coperte da altre frequenze con picco di intensità molto forte. Di conseguenza, queste frequenze possono essere eliminate tranquillamente senza eccessiva perdita di qualità del file.

Il tipo di compressione utilizzato dal formato MP3 è molto efficace: il file originale può essere compresso fino ad una decina di volte della dimensione originale. Un brano stereo, che in formato Wave o AIFF "peserebbe" circa 37 Mb, in MP3 potrebbe scendere alle dimensioni di circa 3 Mb: il risparmio è incredibile, soprattutto se si pensa che in termini di qualità la differenza tra i due file può essere riconosciuta solo da persone abituate ad ascoltare la musica.

Ovviamente, dato che il formato MP3 permette di utilizzare rapporti di compressione diversi e ottenere file di varia qualità e dimensione, compressioni molto elevate generano file di qualità non eccelsa.

Un'altra caratteristica interessante dei file MP3, che li distingue ulteriormente dai formati classici, è rappresentata dalla possibilità di salvare, insieme alla musica, anche altri tipi di informazione, come il nome e l'autore del brano o eventuali commenti, inserendoli in un particolare tag, detto *ID3*. Inoltre, il formato Mp3 è di tipo *stream*, quindi è particolarmente adatto per Internet: i file Mp3 possono dunque essere ascoltati man mano che vengono scaricati dalla Rete, con tutti i vantaggi di cui abbiamo parlato.

5.2.3 Altri formati per Internet

L'MP3 non è l'unico formato utilizzato in Internet. Per diverso tempo si sono utilizzati i file MIDI (*Musical Instrument Digital Interface*), i quali, seguendo un insieme di regole molte complesse, consentono la comunicazione reciproca ed inequivocabile di due strumenti elettronici, come ad esempio due tastiere. I file MIDI (estensione .mid), hanno il vantaggio di essere incredibilmente leggeri: sono infatti, a tutti gli effetti, file di testo e riportano solo le informazioni relative alle modalità di esecuzione del file e non quelle relative all'audio stesso (che richiedono più memoria). La qualità di riproduzione del file dipende dalla scheda audio del singolo computer ma, in generale, è piuttosto bassa.
Un formato invece ancora molto diffuso, la cui popolarità è paragonabile a quella dell'MP3, è RealAudio (estensione .ram); la loro qualità è inferiore a quella del CD, ma resta comunque elevata. Questo formato viene utilizzato principalmente nelle Web Radio.

5.2.4 Formati Video

Un filmato non è che una serie di fotogrammi, riprodotti di seguito in modo così veloce che il nostro occhio non si accorge delle singole immagini, ma percendo il tutto in modo fluido.
Più precisamente, un filmato può essere identificato proprio dal numero di fotogrammi che passano nell'unità di tempo: nel caso della televisione, i fotogrammi al secondo sono 25.
Quando si digitalizza un filmato occorre memorizzare una quantità enorme di informazioni: quelle relative ad ogni singola immagine e, spesso, anche quelle relative ad una traccia audio. I file sarebbero di dimensioni troppo grandi non solo per Internet, ma anche per la riproduzione in locale, in quanto richiederebbero una velocità di trasferimento dati talmente alta da mettere in difficoltà i comuni processori. Nel digitalizzare un filmato si ricorre, quindi, ad un algoritmo di compressione che riduce drasticamente la quantità di informazioni da trattare.
Nel caso dei video, possiamo individuare due tipi di compressione:

- la compressione **spaziale**, che agisce sul singolo fotogramma. Può essere paragonata a quella utilizzata sulle immagini e può essere lossy, ossia con perdita di informazioni, o lossless, ovvero senza tale perdita; questa compressione riduce quindi la quantità di dati richiesta per ogni fotogramma

- la compressione **temporale** che agisce su una serie di fotogrammi: onde evitare inutili ripetizioni di informazioni nel tempo ne analizza aspetti comuni ed elimina le informazioni non necessarie.

5.2.5 Codec

I CODEC sono gli algoritmi di compressione e decompressione dei file e vengono categorizzati in due modi diversi: a seconda del tipo di compressione impiegata e come *simmetrici* o *asimmetrici*:

- *simmetrici* quando il processo di compressione e decompressione avviene in tempi uguali
- *asimmetrici* quando il processo di compressione e decompressione avviene in tempi diversi.

Uno stesso formato video può utilizzare diversi tipi di CODEC.

5.3 Darwin Streaming Server

Darwin Streaming Server è il server che abbiamo utilizzato nell'ambito di questo progetto. Darwin è la versione open-source di QuickTime Streaming Server di Apple e condividono stesso codice di base. DSS permette di inviare contenuti audio video su Internet utilizzando i protocolli standard RTP e RTSP e permette un elevato livello di personalizzazioni e di controllo delle prestazioni.
Darwin Streaming Server può essere eseguito su diverse piattaforme come Windows Linux e Solaris.

5.3.1 Formati multimediali supportati

Il server Darwin richiede che i formati dei file multimediali da inviare su Internet siano compressi e "hinted". Rendere hinted un file significa, infatti, utilizzare particolari software che analizzino i dati e creino delle tracce in cui vengano specificate varie informazioni necessarie al server per "pacchettizzare" i dati stessi da trasmettere. In un hinted file si troveranno allora tante tracce quante sono le tracce nel media originale a cui saranno allegati dettagli sul tipo di codifica utilizzato, bit rate, frame rate se video o frequenza di campionamento se audio, durata della traccia e informazioni esplicite su dove

sono localizzati i payload delle varie tracce; la traccia “hint” non viene mai inviata attraverso la rete. La compressione è necessaria per le tracce video in modo particolare perché la diffusione su Internet richiederebbe centinaia di megabit per secondo per ogni singolo file in streaming. Perciò, ogni file viene compresso tramite un codec quando deve essere inviato e successivamente quando deve essere eseguito viene decompresso utilizzando sempre lo stesso codec.
I formati e i codec audio supportati da Darwin Streaming Server sono riportati nella tabella successiva.

FILE FORMAT:	**VIDEO CODECS SUPPORTED INCLUDE:**
.mov	Cinepak, H.261, H.263, MPEG-4 Video, Sorenson Video, Sorenson Video 3, Video
.mp4	MPEG-4 Video
.3gp	MPEG-4 Video

Tabella 5.2 - Codec video supportati dal server

Anche per le tracce audio possono trovare beneficio dalla compressione per effettuare lo streaming e nella tabella successiva vengono evidenziati i formati e i codec che il server è in grado di inviare correttamente attraverso Internet:

FILE FORMAT:	**AUDIO CODECS SUPPORTED INCLUDE:**
.mov	IMA 4:1, MPEG-4 Audio (AAC Low Complexity), QDesign Music 2, Qualcomm PureVoice, uLaw 2:1
.mp4	MPEG-4 Video (AAC Low Complexity)
.3gp	AMR-NB (Speech), AAC Low Complexity

Tabella 5.3 - Codec audio supportati dal server

5.3.2 Amministrazione del server

Una delle caratteristiche più importanti di questo server è la possibilità di effettuare l’amministrazione del server da remoto utilizzando un web-browser (vedi figura 5.1).
Questo rende possibile la risoluzione di molti problemi e di effettuare un controllo sugli accessi e sulle risorse anche senza intervenire sul server stesso.

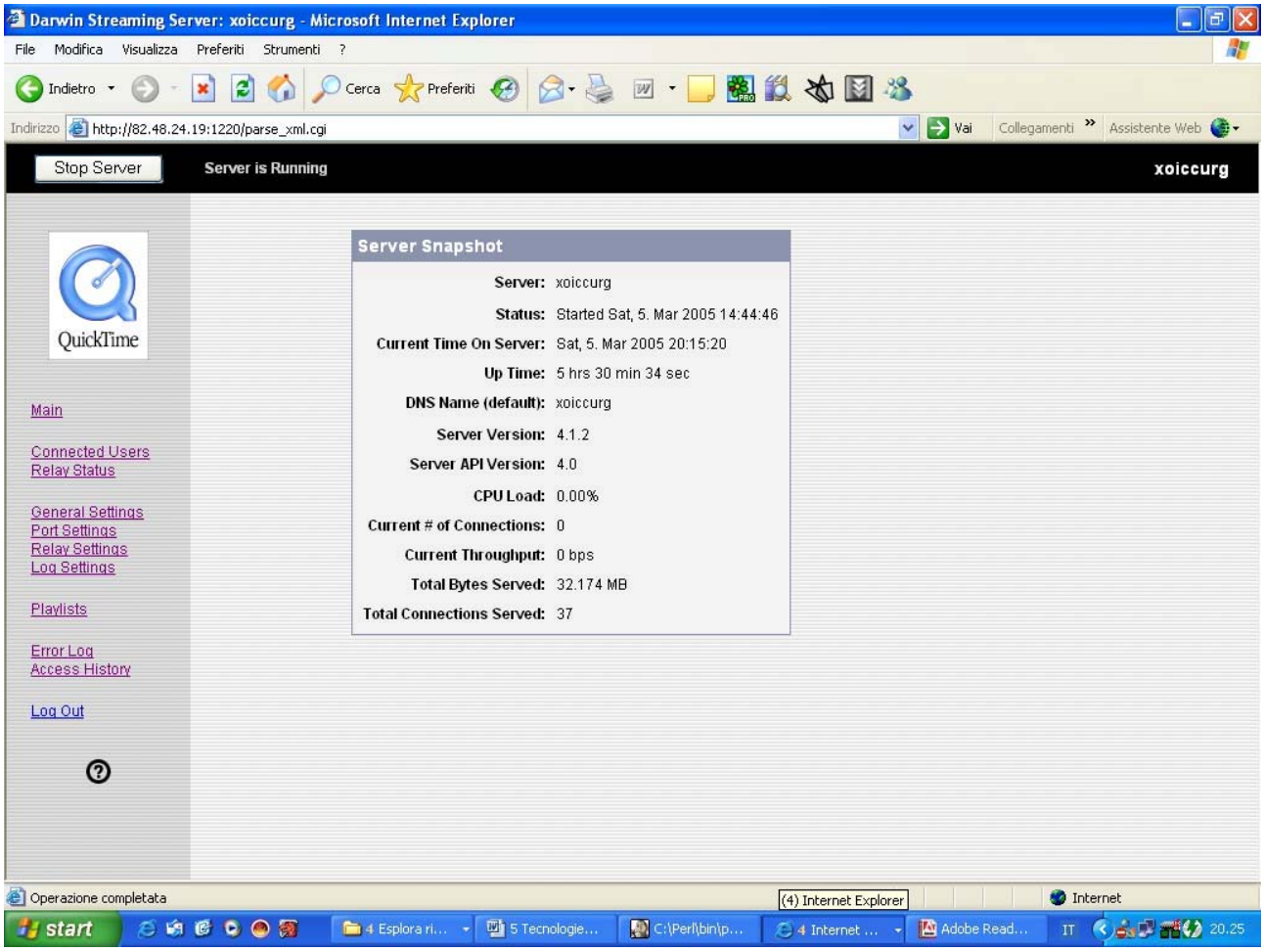

Figura 5.1 - Amministrazione Darwin Streaming Server

In particolare diamo uno sguardo ad alcune delle caratteristiche di amministrazione che possono essere monitorate e configurate da remoto sul server:

- Start e Stop del server
- Monitoraggio degli utenti connessi. In questa sezione si possono ottenere diverse informazioni sull'utente connesso quali l'IP della macchina, il tempo di connessione e il file al quale l'utente è interessato. Inoltre vengono forniti i valori di bit rate e di i byte totali inviati per quel file e la percentuale di pacchetti persi.
- Gestione delle opzioni di amministrazione. Viene data la possibilità di modificare le password di amministrazione e di gestione delle playlist; inoltre si possono settare altre opzioni quali il numero massimo di utenti connessi, il massimo throughput e il tipo di autenticazione che si vuole utilizzare.
- Gestione dell'invio di dati sul protocollo HTTP. Il server può effettuare lo streaming utilizzando i protocolli RTSP/RTP e mentre RTSP invia messaggi

attraverso TCP, RTP, come già discusso nel secondo capitolo, invia i dati utilizzando UDP. Molti firewall però sono configurati per lasciar passare i pacchetti TCP in base al numero di porta e non permettono o limitano il traffico di UDP. Per poter comunque effettuare lo streaming esistono quindi tre differenti possibilità:

1. Effettuare lo streaming sulla porta 80. Questa opzione permette allo streaming server di incapsulare il traffico RTSP e RTP all'interno di pacchetti TCP inviati alla porta 80 che è la porta predefinita per il traffico ed è generalmente lasciata aperta dai firewall. Questo tipo di approccio comporta la riduzione delle performance della rete e richiede connessioni lato client molto veloci per mantenimento dello streaming. Inoltre comporta un maggior carico per il server
2. Apertura di un opportuna porta nel firewall. Questo permette al server di essere accessibile attraverso RTSP e RTP attraverso la porta predefinita e permette un miglior uso delle risorse di rete. Inoltre richiede una connessione lato client non eccessivamente veloce e comporta un minor carico per il server.

   Le porte che devono essere aperta per avere uno streaming di questo tipo sono:

   porta TCP 80 che viene usata per la segnalazione RTSP al server, la porta TCP 554 utilizzata da RTSP e le porte UDP dalla 6970 alla 9999 (anche se viene tipicamente usato un campo più ridotto di porte 6970-6999) che servono per lo streaming RTP.
3. Creazione di un Proxy Server. Il proxy viene posto tra un firewall esterno e il firewall interno di una rete locale. Utilizzando le regole del firewall, i pacchetti che vengono inviati alle porte descritte precedentemente vengono inoltrate dal proxy attraverso il firewall interno al client e dal proxy su Internet attraverso il firewall esterno. Al client non viene permesso di creare una connessione diretta alla risorsa tramite queste porte ma tutto avviene attraverso il proxy server che rappresenta un livello aggiuntivo per la sicurezza.

- Gestione e monitoraggio del Relay.Viene utilizzato per la distribuzione del carico sui vari server creando un uso più efficiente della rete

- Gestione e visualizzazione dei file di log.
- Creazione e gestione delle playlist
- Visualizzazione delle informazioni sulle richieste dei contenuti multimediali

5.4 Strumenti e tecnologie JAVA

Per poter sviluppare ed analizzare un progetto basato sulla Java Media Framework abbiamo utilizzato un ambiente IDE di sviluppo ed abbiamo scelto Eclipse nella versione 3.0.1 per la sua facilità di utilizzo e di gestione dei progetti e del codice sorgente, in particolare il codice sorgente della JMF 2.1.1e. Inoltre si sono utilizzate le librerie della JAVA2SE nella versione 1.4.
Per poter creare un applicazione funzionante è importante evidenziare i formati e i codec audio e video supportati dal JMF per quanto riguarda la ricezione tramite RTP che vengono riportati nelle seguenti tabella:

MEDIA TYPE	RTP PAYLOAD	JMF 2.1.1 CROSS PLATFORM VERSION	JMF 2.1.1 SOLARIS/LINUX PERFORMANCE PACK	JMF 2.1.1 WINDOWS PERFORMANCE PACK
Audio: G.711 (U-law) 8 kHz	0	R,T	R,T	R,T
Audio: GSM mono	3	R,T	R,T	R,T
Audio: G.723 mono	4	R	R,T	R,T
Audio: 4-bit mono DVI 8 kHz	5	R,T	R,T	R,T
Audio: 4-bit mono DVI 11.025 kHz	16	R,T	R,T	R,T
Audio: 4-bit mono DVI 22.05 kHz	17	R,T	R,T	R,T
Audio: MPEG Layer I, II	14	R,T	R,T	R,T
Video: JPEG (420, 422, 444)*	26	R	R,T	R,T
Video: H.261	31	-	R	R
Video: H.263**	34	Mode A Only	R,T	R,T
Video: MPEG-I***	32	T	R,T	R,T

Tabella 5.4 - Formati e codec supportati dalla JMF con RTP (R indica che il formato può essere decodificato e presentato, T indica che il formato può essere codificato e trasmesso, ** H263/RTP può essere trasmesso in tre differenti dimensioni: SQCIF (128x96), QCIF (176x144) e CIF (352x288)

5.5 Preparazione del file multimediale per lo streaming: QuickTime Pro

QuickTime Pro è un programma commerciale sviluppato da Apple in grado di riprodurre diversi formati multimediali e di convertire i file in altri formati. Proprio questa caratteristica è stata utile nello sviluppo di questa tesi per creare dei file da utilizzare per effettuare lo streaming attraverso il server Darwin e riproducibili attraverso un applicazione creata con l'ausilio di Java Media Framewrok.
Vediamo ora quali sono le opzioni principali per convertire un file con questo programma.
Per prima cosa bisogna aprire un file dal menu "open" e quindi sempre dal menu "file" selezionare l'opzione "export". Si apre ora una finestra di salvataggio del file e ciccando sul pulsante "options" si accede alla finestra riportata in figura.

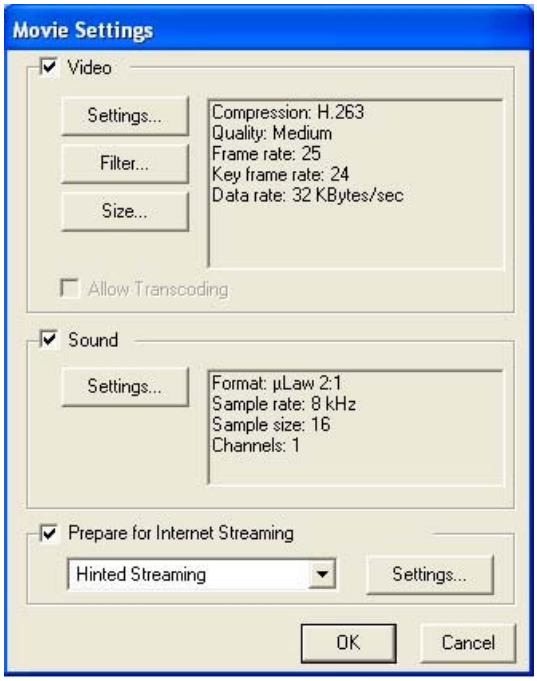

Figura 5.2 - QuickTime Pro opzioni per la conversione del filmato

Nella sezione video è possibile selezionare tre tipi di impostazioni:

- "settings": si può selezionare il codec, la qualità video, il numero di frame per secondo ed impostare il numero di "key frame" ed il limite di "data rate"
- "filter": in questa sezione possono essere aggiunti filtri di varia natura
- "size": serve per impostare una dimensione personalizzabile del file

Nella sezione sound le opzioni riguardano:

- il tipo di compressione
- la frequenza di campionamento
- i bit di campionamento
- e la modalità stereo o mono

Infine nell'ultima sezione si possono scegliere le opzioni per effettuare lo streaming su Internet del file. In particolare per i file "hinted" c'è la possibilità di ottimizzare il file per il server e di specificare il tipo di codifica e di pacchetto per i payload RTP.

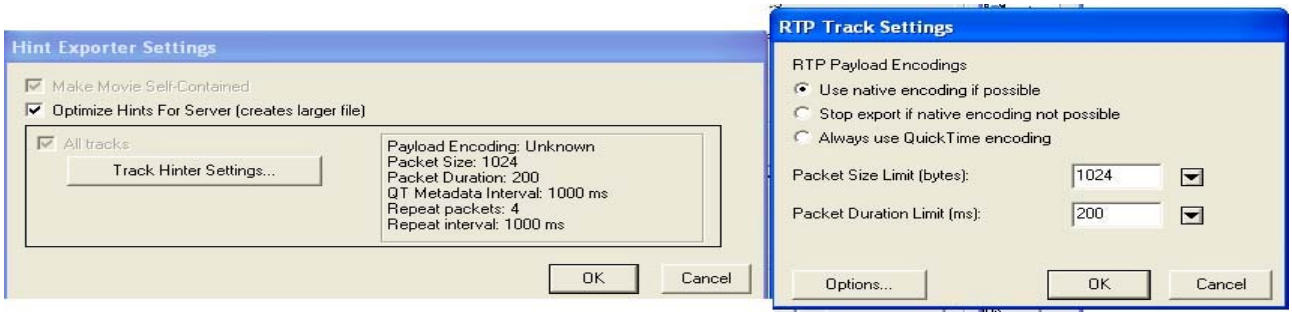

Figura 5.3 - Opzioni per la conversione di un file "hinted"

5.6 Ethereal

Ethereal è un analizzatore di protocollo open source, rilasciato sotto licenza GNU GPL, disponibile per diverse piattaforme e sistemi operativi, nella figura successiva osserviamo come si presenta l'interfaccia del programma.
Rappresenta uno strumento essenziale per ogni amministratore di rete, perché permette di osservare ogni tipo di pacchetto che viene inviato o ricevuto su un computer collegato alla rete.
Nell'ambito di questa tesi è servito per analizzare lo scambio di messaggi RTSP tra server e client e per osservare come vengono inviati i dati audio video attraverso RTP e le informazioni di controllo di RTCP.

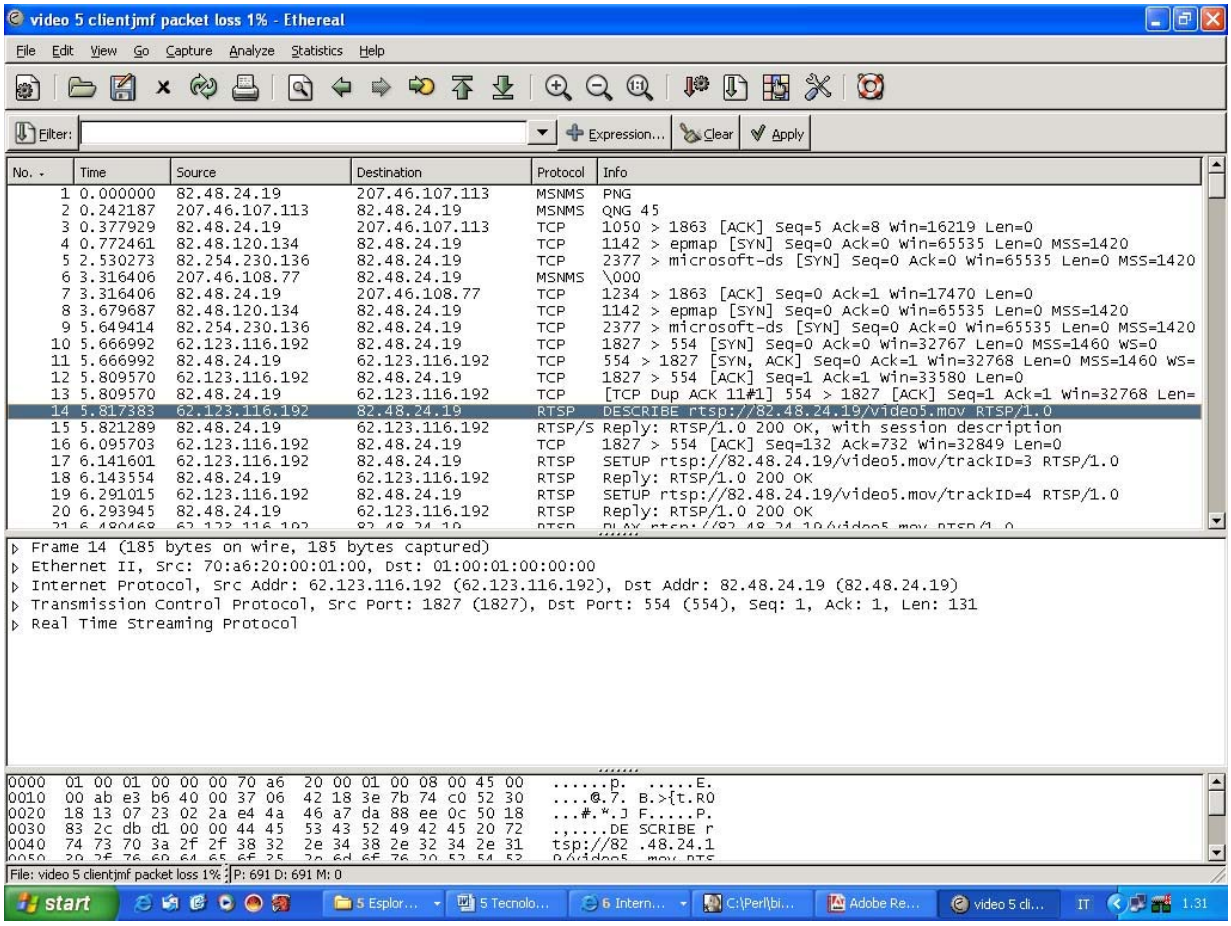

Figura 5.4 - Interfaccia di Ethereal

Si noti come in Ethereal in pacchetti in ingresso e in uscita vengano suddivisi in base al tipo di protocollo del pacchetto. Questa operazione di “parsing”, che si può notare anche nella figura precedente, aiuta in modo sostanziale nell’analisi dei messaggi RTSP.

6 Analisi e Progetto

6.1 Introduzione

In questa tesi, come già accennato nel capitolo precedente, ci si propone di studiare le librerie offerte dalla Java Media Framework per la creazione di player multimediali che realizzino un client per il protocollo RTSP.
In questo capitolo vedremo un analisi della libreria JMF riguardante il protocollo RTSP che è stata creata attraverso un lavoro di "*reverse engineering*" per sopperire alla mancanza di documentazione su questa parte della libreria.
In seguito vedremo come è stata creata l'applicazione e in particolare i metodi che permettono di utilizzare il supporto fornito dalla libreria per la creazione di un player multimediale che riproduca lo streaming ricevuto da un server RTSP e i metodi per gestire eventuali errori di connessione con il server.
Infine saranno presentati alcuni test sull'applicazione.

6.2 Analisi dei Requisiti

Il sistema che vogliamo realizzare si pone nel campo dei servizi multimediali forniti attraverso la rete dove non sempre l'interoperabilità tra servitore e cliente è garantita se non per particolari architetture costruite "su misura". In particolare si vuole realizzare un client possa connettersi ad un server che supporti il protocollo RTSP. Proprio il supporto a RTSP, ad altri protocolli per la comunicazione (RTP e RTCP) e a metodi per la descrizione della sessione come SDP permettono la comunicazione fra client e server di diversi.

6.3 Analisi libreria JMF per il protocollo RTSP

Analizziamo in quale modo il framework gestisce la richiesta di creazione di un client RTSP che avviene attraverso il metodo *Manager.createPlayer* che come abbiamo sottolineato rappresenta il punto centrale e fondamentale per la creazione di un player RTSP.

In particolare le principali classi che si occupano della creazione e gestione del player RTSP sono:

- *Manager*
- *DataSource*
- *Handler*
- *RtspUtil*
- *RtspManager*
- *Connection*

All'interno del package com.sun.media.rtsp.protocol la JMF offre delle classi per:

- creare i messaggi per i metodi: DESCRIBE, OPTIONS, PAUSE, PLAY, SET_PARAMETER, SETUP e TEARDOWN
- creare gli header: ContentBase, Cseq, Duration, Range, Session, Trasport
- creare messaggi di Request e Response con relative Status Line e Status Code

6.3.1 Classe *Manager*

La classe *Manager* è quella che permette la creazione del player in base al tipo di sorgente dei dati. Fornisce vari metodi per ottenere dei player; riportiamo in particolare quelli che vengono utilizzati per la creazione di un Player RTSP:

- *createPlayer*: è il metodo che si occupa di riconoscere il tipo di protocollo utilizzato e verifica la necessità di utilizzare un plugin per la costruzione del player. Richiama il metodo *createPlayerForContent*.
- *createPlayerForContent*: si crea un vettore di possibili *DataSource* che possono implementare il client e si tenta mediante la reflection di istanziare i *DataSource*. Trovato quello adatto si richiama *createPlayerForSource* che fornirà il player
- *createPlayerForSource*: viene creato un vettore contenente i possibili *Handler* per il player e come per il metodo precedente si cerca quello appropriato tentando di istanziare quello adatto tramite la reflection. Trovato l'handler adatto viene assegnato al player che viene quindi restituito.

6.3.2 Classe *DataSource*

Questa classe eredita dalla classe *BasicPushBufferDataSource* ed è l'astrazione per il framework della sorgente di dati che in questo caso è il protocollo RTSP che permette il controllo sul trasporto dei dati dal server RTSP.
Fra i metodi principali ricordiamo il *setLocator* che permette d'impostare il *MediaLocator* contenente l'URL RTSP che vogliamo ricevere e il metodo *getContentType* che restituisce il tipo di protocollo che gestisce questo *DataSource*.

6.3.3 Classe *Handler*

Handler è una classe che eredita da *BasicPlayer* ed implementa le interfacce *ReceivedStreamListener*, *TimerListener* e *BufferListener* e si occupa di gestire il player durante la sessione RTSP utilizzando metodi quali:

- *initRtspSession*: questo metodo gestisce le creazioni della connessione al server, dell'inizializzazione della sessione RTSP, e del manager per RTP con relativo buffer
- *doStart*: richiama il metodo che controlla l'inizio della presentazione via RTSP
- *doStop*: richiama il metodo che controlla l'arresto della presentazione RTSP
- *doClose*: richiama i metodi per la chiusura della connessione e del player

6.3.4 Classe *RtspUtil*

Questa classe implementa l'interfaccia *RtspListener* ed è caratterizzata da molti metodi per la creazione e l'invio dei messaggi tipici di RTSP.
Tra i metodi di maggior importanza ricordiamo:

- *rtspSetup*: si occupa di preparare e di inviare i messaggi DESCRIBE e SETUP al server e crea il *Manger* RTP. Inoltre individua il numero e il tipo delle tracce da riprodurre per permettere una corretta costruzione dei messaggi di SETUP.
- *rtspStart*: prepara il messaggio PLAY per il server e ne richiede l'invio. Gestisce eventuali errori riguardanti questo invio

- *rtspStop*: come il metodo precedente prepara il messaggio PAUSE per il server e ne richiede l'invio gestendo gli errori relativi a questo invio.
- *rtspTeardown*: anche questo come per i due metodi precedenti prepara un messaggio RTSP, in questo caso quello di TEARDOWN, e ne richiede l'invio gestendo gli errori di timeout che possono essere generati
- *rtspMessageIndication*: permette di suddividere i messaggi in arrivo tra quelli di richiesta che vengono poi gestiti da *processRtspRequest* e quelli di risposta che vengono gestiti da *processRtspResponse*
- *getStatus* e *getTextStatus* permettono rispettivamente di ricevere il numero di codice di stato ricevuto nel messaggio di risposta e l'indicazione del testo del codice di stato
- *sendMessage* richiama i metodi per l'invio dei messaggi verso il server
- *sendResponse* prepara il messaggio di risposta e richiede l'invio tramite il metodo *sendMessage*

6.3.5 Classe *RtspManager*

La classe *RtspManager* si occupa dell'apertura e della chiusura delle connessioni ed effettua l'invio dei messaggi. In particolare i metodi dedicati a queste operazioni sono:

- *createConnection*: questo metodo crea un oggetto di tipo *Connection* e lo aggiunge ad una lista di eventuali connessioni aperte. Vengono generati gli errori dovuti a problemi nella connessione con il server
- *closeConnection*: questo metodo serve per chiudere la connessione e quindi liberare le risorse da essa occupate
- *sendMessage*: verifica che la connessione esista e invia i dati utilizzando il metodo *sendData* della classe *Connection*

6.3.6 Classe *Connection*

La classe Connection eredita dalla classe *Thread* e implementa l'interfaccia *Runnable*. Questa classe crea una *Socket* per la connessione al server e fornisce i seguenti metodi per lo scambio di dati con il server:

- *run*: se la connessione è attiva riceve dati dal server utilizzando un oggetto *MediaProcessor* che tramite il metodo *processMessage* permette il passaggio di responsabilità per i messaggi in arrivo al metodo *rtspMessageIndication* della classe *RtspUtil*
- *sendData*: si occupa dell'invio dei dati tramite la socket al server.
- *cleanup*: si occupa della chiusura della socket e quindi della connessione

6.4 Progetto

L'applicazione che dobbiamo realizzare è un client RTSP che richiede ad uno streaming server un particolare contenuto multimediale utilizzando i protocolli RTSP RTP e RTCP.
Per la realizzazione di un client di questo tipo è necessario creare una connessione al server, gestire i comandi e gli header RTSP (almeno quelli fondamentali per l'inizializzazione, la riproduzione e la terminazione del flusso di dati quali: SETUP, PLAY e TEARDOWN), gestire il flusso di dati in arrivo dal server tramite il protocollo RTP. I dati spediti dal server devono essere presentati in maniera opportuna ad esempio utilizzando un player differente a seconda che i dati ricevuti siano di tipo audio o di tipo video. Inoltre devono venire gestiti i possibili errori di richiesta della risorsa tra i quali ad esempio: l'errore di immissione dell'indirizzo del server o del nome del file. Devono anche essere gestiti gli errori del server che vengono segnalati tramite risposte RTSP o errori di timeout utili per non dover aspettare indefinitamente una risposta. Tutti gli errori devono essere presentati all'utente nella maniera più opportuna e più chiara possibile in modo che sia chiaro quale sia la causa di tale malfunzionamento.

6.5 Implementazione

L'applicazione si basa sull'utilizzo di diverse classi per la creazione dell'interfaccia grafica dove inserire il player multimediale che è la parte centrale dell'applicazione. La creazione del player avviene richiedendo alle librerie della JMF l'opportuno player per un URL RTSP. Il player viene costruito dentro a un frame per poterlo poi visualizzare e riprodurre. Nello schema successivo diamo un idea dell'interazione tra le classi dell'applicazione:

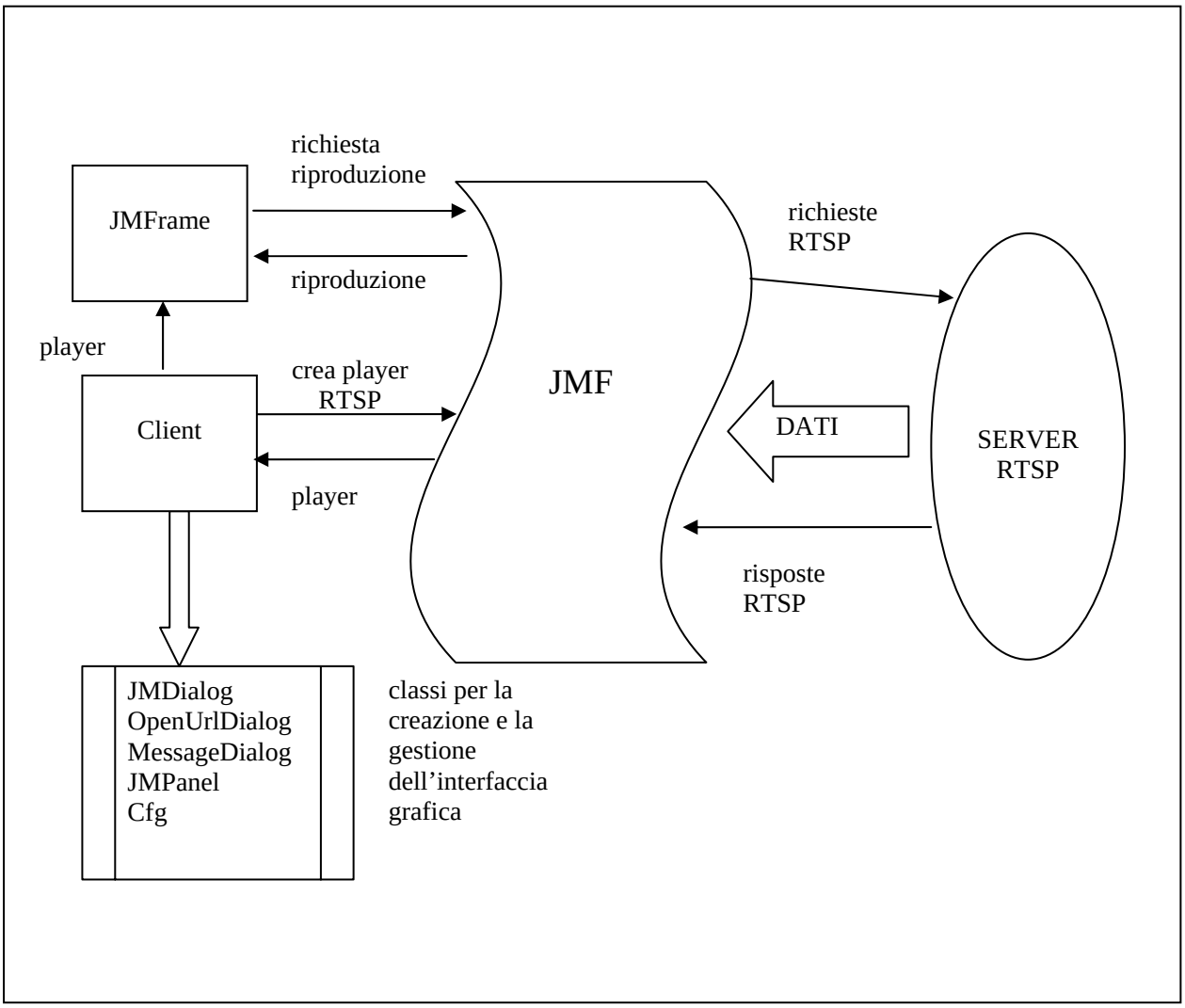

Figura 6.1- Astrazione del funzionamento dell'applicazione

Ora presentiamo brevemente le classi create ed utilizzate nel progetto:

- *Client*: è insieme alla classe JMFrame la classe principale dove vengono richiamati i metodi per la creazione del player e la gestione degli errori relativi al collegamento con il server. Inoltre vengono costruiti e gestiti i menu dell'applicazione. Eredita dalla classe *Frame* e implementa l'interfaccia *ControllerListener*
- *JMFrame*: eredita dalla classe *JInternalFrame* e implementa l'interfaccia *ControllerListener*. Fornisce il supporto per creare un frame con un lettore

multimediale che viene creato in modo differente in base al tipo di contenuto da riprodurre. Gestisce gli eventi generati dal framework per la riproduzione del contenuto multimediale.

- *JMDialog*: fornisce dei metodi per la creazione la disposizione e la chiusura di finestre di dialogo. Eredita dalla classe *Dialog* ed implementa le interfacce *ActionerListener* e *WindowListener*
- *OpenUrlDialog*: eredita dalla classe *JMDialog* e permette la creazione di una finestra per l'inserimento dell'URL per ottenere la risorsa multimediale indicata
- *MessaggeDialog*: fornisce metodi per la creazione di finestre contenenti messaggi di errore.
- *JMPanel*: eredita dalla classe Panel e attraverso i suoi metodi permette la formattazione degli oggetti di questo tipo
- *Cfg*: permette il salvataggio dell'ultimo URL immesso nella maschera "Open URL" se l'applicazione non viene chiusa

Concentriamoci sulle parti di codice più significative per l'implementazione del client. In particolare vediamo in che modo deve essere creato il client utilizzando il metodo *Manager.createPlayer*.

Nella classe *Client* viene costruito un oggetto *MediaLocator* utilizzando il nome dell'URL, del contenuto multimediale che vogliamo visualizzare, come argomento necessario per creare questo oggetto. Per evitare la creazione di un oggetto non valido controlliamo il contenuto dell'oggetto appena creato e nel caso il *MediaLocator* non venga creato correttamente si segnala l'errore e il metodo viene terminato. Se invece, tutto è andato a buon fine si cerca di creare un oggetto *Player* utilizzando il metodo *Manager.CreatePlayer*. Si noti come sia proprio questo il metodo fondamentale per la realizzazione di un client RTSP perché viene passato come argomento al metodo per la creazione del player un URL RTSP e le librerie della JMF, come abbiamo visto nell'analisi, gestiscono attraverso varie classi questo tipo di creazione preparando la connessione al server e gestendo in maniera opportuna i vari protocolli di comunicazione con il server.

```
mrl = new MediaLocator ( nameUrl );
if ( mrl == null  ||  nameUrl.equals("") )
{
Fatal ("Can't build URL for" + " " + nameUrl);
return;
}
try {
player = Manager.createPlayer(mrl);
} catch (NoPlayerException e) {
Fatal("Could not create player for " + mrl);
return;
}
if (player != null) {
player.addControllerListener(this);
JMFrame jmframe = new JMFrame(player, nameUrl);
desktop.add(jmframe);
}
```

Viene inoltre creato un nuovo oggetto JMFrame per aggiungere il frame e quindi il player in esso contenuto all'applicazione.
I seguenti metodi utilizzati sul player portano il player in uno stato differente fino alla riproduzione vera e propria che viene ottenuta utilizzano il metodo *start*. Il metodo *close* chiude il player e rilascia le risorse a lui associate.

```
player.realize();
player.prefetch();
player.start();
player.close();
```

Per ottenere un'applicazione semplice ma completa non si può prescindere dalla generazione di messaggi di errore per segnalare la presenza di problemi nella connessione al server. Per realizzare queste operazioni si è scelto di creare la classe *Client* come implementazione di *ControllerListener* e di aggiungere il player alla lista di "ascoltatori" di *ControllerListener*.

Questo ci permette, in caso di evento di tipo *ControllerEventError*, di catturare l'evento e gestirlo segnalando in modo chiaro all'utente l'errore e interrompendo la creazione del player.

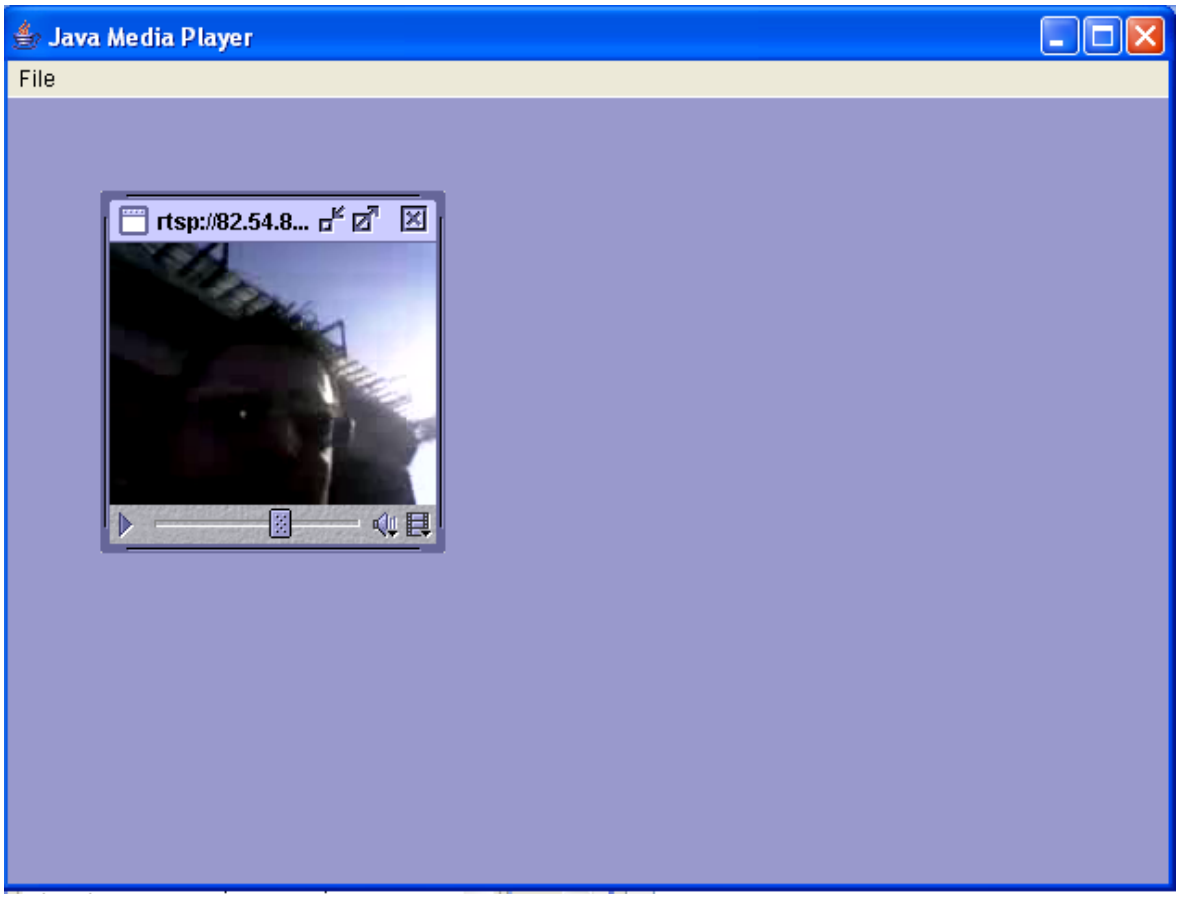

Figura 6.2 - Layout del client

Il layout dell'applicazione è quello mostrato nella figura precedente e si presenta come una semplice finestra dotata di un menu dal quale può essere selezionata e richiamata l'opzione "Apri URL".
Selezionando questa opzione si apre un una maschera dove possiamo inserire l'URL ad esempio rtsp://82.54.81.19/video.mov. Se l'indirizzo del server è corretto e il file cercato esiste, si aprirà il lettore multimediale riproducendo il contenuto richiesto, altrimenti verrà generata una finestra che segnalerà il tipo di errore e il lettore non viene caricato.

6.6 Test

I test effettuati sull'applicazione e sulle classi della JMF sono stati di due tipi:

1. confronto dello scambio dei messaggi tra client e server utilizzando come client QuickTime Pro e l'applicazione realizzata con la JMF.
2. monitoraggio del bit rate e dei pacchetti persi effettuato tramite le utilità del server utilizzando come client QuickTimePro e il client che utilizza JMF

Tutti i test sono stati effettuati con il server Darwin nella versione 4.0.2 installato su una macchina con le seguenti caratteristiche: AMD 64 3400+ RAM 1G connessione ad internet 1280/256 Kbit/s. Entrambi i client su un PENTIUM 4 3GHz RAM 512MB connessione ad internet 640/256 Kbit/s.
Nei test è stato utilizzato un file audio video "hinted" con codifica H.263 per il video e codifica G.711 (U-law) 8KHz mono della durata di circa 30 secondi.

6.6.1 Test 1

Il secondo tipo di test prevede l'analisi dei pacchetti di controllo e di dati tra il server e il client catturati tramite il software Ethereal presentato nel capitolo precedente.
Riportiamo nella figura seguente i tipi di messaggi scambiati per entrambi il client:

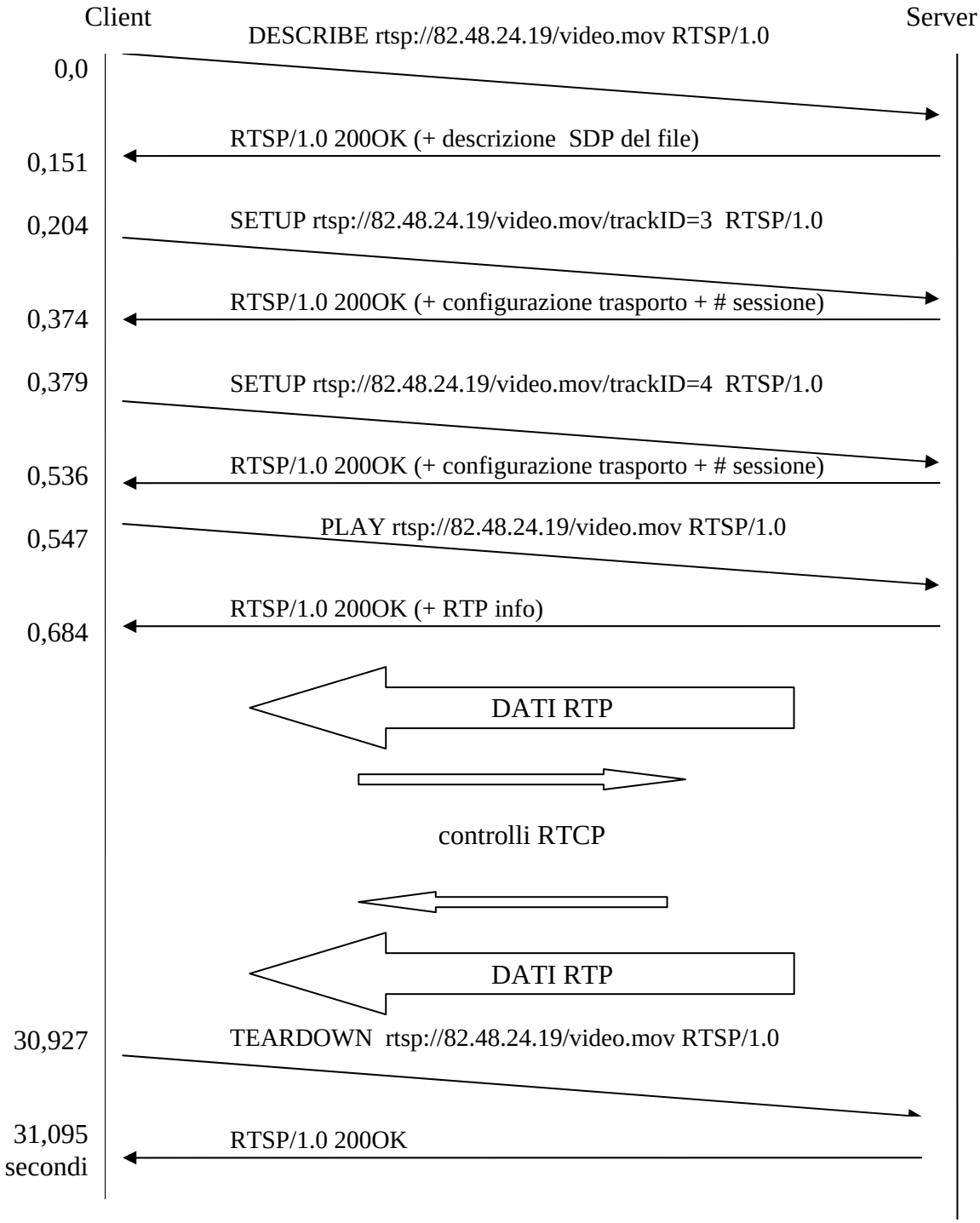

Figura 6.2 - Scambio di dati tra client e server

Sia il client QuickTime Pro che il client realizzato tramite la JMF inviano e ricevono questi messaggi. Il client QuickTime Pro invia anche un comando RTSP OPTIONS * RTSP/1.0 a cui il server risponde con un messaggio di errore “501 Not Implement” perché non supporta quel comando.

Il comando DESCRIBE richiede la descrizione al server di un file e il server invia insieme alla risposta contenente lo stato della risposta un file di descrizione SDP con le informazioni principali riguardanti il contenuto multimediale da noi richiesto. Tra le informazioni principali vi sono: nome della sessione, durata totale della sessione (attributo range), informazioni sulla banda minima per effettuare il playback del file, e informazioni sulla traccia audio e video, che utilizzando un tipo di payload dinamico viene segnalato nel campo “rtpmap”.

Il comando SETUP viene ripetuto per ogni traccia del file multimediale da riprodurre perché il server non supporta il controllo aggregato e specifica le il tipo di trasporto che vuole utilizzare esplicitando le porte sulle quali vuole ricevere lo stream. Il server accetta le richieste di SETUP e invia le risposte indicando le porte dalle quali invierà lo stream.

Il comando di PLAY richiede l’inizio dello streaming e il server risponde aggiungendo il l’header “RTP-info” contente i campi “url”, “seq” e “rtptime” necessari al client per effettuare la sincronizzazione.

I flussi di dati audio e video vengono inviati tramite RTP sulle porte negoziate precedentemente tra il client e il server. In particolare il client specifica quattro porte: due per la ricezione dei dati attraverso RTP per suddividere la ricezione del flusso audio da quello video e due per la ricezione e l’invio di messaggi RTCP (una per l’audio, l’altra per i dati video). Il server invece utilizza solo due porte e con una effettua l’invio dei dati mentre con l’altra invia e riceve i rapporti RTCP.

La comunicazione termina quando il client utilizza il comando di TEARDOWN e il server che lo riceve invia la risposta al client di accettazione di chiusura della connessione. Prima della chiusura della connessione vengono inviati i rapporti sulla qualità del servizio tramite RTCP.

Il server supporta anche il comando RTSP di PAUSE che funziona correttamente dando la possibilità di ripartire da punto in cui si era fermato il clip utilizzando opportunamente l’header “range”.

Si sottolinea come tutti i comandi RTSP vengono inviati tramite TCP mentre tutti i dati e i controlli sulla qualità RTP-RTCP vengono inviati tramite UDP proprio come ci aspettavamo.

6.6.2 Test 2

Questo test è rivolto alle prestazioni dei client QuickTime Pro e del client che utilizza JMF. Il test è stato ripetuto richiedendo per cinque volte consecutive lo stesso file ai due client:

Bit rate (Kbit/s) a regime	% pacchetti persi (picco)	Mem server (Mbyte)	Mem Client (Mbyte)
243	0	4,144	12,1
249	0	4,144	12,4
253	0	4,184	12,4
250	0	4,184	12,4
250	0	4,184	12,5

Tabella 6.1 - QuickTime Pro

Bit rate (Kbit/s) a regime	% pacchetti persi (picco)	Mem server (Mbyte)	Mem Client (Mbyte)
178	1,35	4,180	21,7
178	1,37	4,184	21,7
178	1,40	4,180	22,0
178	1,35	4,184	22,5
178	1,35	4,180	22,5

Tabella 6.2 - Client JMF

Dalle tabelle si può osservare come il bit rate del client QuickTime Pro sia decisamente superiore consentendo una migliore qualità nella riproduzione video.
Si sottolinea come la percentuale di pacchetti persi da parte del client JMF avvenga solamente nei primi istanti di connessione al server e che successivamente non vi è più perdita di pacchetti. La perdita di pacchetti è dovuto al buffering della prima parte della riproduzione video.
La memoria occupata durante l'utilizzo del client JMF è nettamente superiore per via dell'utilizzo della Java Virtual Machine e non di componenti nativi per la creazione del client. Grazie a questo il client JMF può facilmente essere usato su macchine con sistemi operati diversi da quello su cui è stato creato.

6.7 Conclusioni

L'analisi delle classi della JMF ci ha permesso di scoprire quali siano i comandi e gli header RTSP supportati dal framework. Inoltre grazie ai test si è verificato il buon comportamento in presenza di errori da parte dell'applicazione che utilizza JMF e il puntuale utilizzo dei comandi principali nella connessione al server. Utilizzando le classi del package com.sun.media.rtsp.protocol possono essere anche implementati OPTIONS e SET_PARAMETER non visti nei test.

Il server utilizzato è stato Darwin Streaming Server nella versione 4.0.2 ma è disponibile una versione la 5.0.1 che però non permette la corretta connessione con la JMF per un problema di invio anticipato dei pacchetti RTP di dati prima della risposta alla richiesta di PLAY. La Modifica di alcune classi del JMF dovrebbe risolvere il problema.

Conclusioni

Durante il lavoro per la redazione della tesi abbiamo studiato vari protocolli per la comunicazione multimediale, abbiamo compreso le problematiche legate allo sviluppo di servizi multimediali attraverso la rete capendo l'importanza della separazione dei protocolli per la consegna effettiva dei dati, da quelli necessari per il controllo dei flussi e della sessione. Nella parte sperimentale abbiamo studiato la comunicazione tra un client e un server RTSP mediante l'utilizzo di un software (Ethereal) in grado di effettuare lo "*sniffing*" dei pacchetti in transito sul sistema, che ci ha permesso di osservare da un punto di vista pratico gli aspetti più rilevanti dei protocolli precedentemente studiati. Questo aspetto è stato molto utile, in particolare è stato importante per chiarire i dubbi legati al solo studio teorico.

Per la realizzazione del client si è studiato ed utilizzato il Java Media Framework (JMF), un framework che è in grado di fornire un supporto per la creazione di componenti multimediali per applicazioni di rete. In particolare, è stato realizzato un componente che permette la connessione e l'interazione con server di streaming che erogano servizi multimediali.

Abbiamo compreso pregi e limiti del framework ed abbiamo capito in quale modo supporti la creazione di applicazioni per la ricezione di contenuti multimediali.

Nella fase di test abbiamo potuto vedere come il client creato utilizzando JMF supporti i comandi tipici di RTSP permettendo una buona comunicazione con il server e realizzando l'obbiettivo che ci eravamo prefissi, cioè la possibilità di utilizzare componenti non creati "*ad hoc*" per comunicare tra loro attraverso il protocollo RTSP.

Possibili linee di sviluppo future includono l'interazione di un client che utilizza JMF con diversi server RTSP, e.g., RealNetworks (Helix) oppure l'allargamento dei formati e dei codec multimediali supportati dal JMF con RTP, che rappresenta uno dei principali limiti di questo framework e che risulta essere uno dei limiti per l'interazione con alcuni tipi di server per la mancanza di formati e codec supportati comuni.

Bibliografia

[APP01] Apple, QuickTime Streaming Server, Darwin Streaming Server Amministrator's Guide

[RFC2326] H. Schulzrinne, A. Rao, R.Lanphier, Real Time Streaming Protocol (RTSP).

[RFC2327] M. Handley, V. Jacobson., SDP: Session Description Protocol.

[RFC1889] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, A Transport Protocol for Real-Time Applications. Audio-Video Transport Working Group

[RFC3261] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler., SIP: Session Initiation Protocol.

[RFC3551] H. Schulzrinne, S. Casner.,RTP Profile for Audio and Video Conferences with Minimal Control.

[RTSP] www.rtsp.org

[MMUSIC] http://www.ietf.org/ids.by.wg/mmusic.html

[CCA] Ralf Steinmetz, Klara Nahrstedt, Mutilimedia: Computing, Communications and Applications

[IMO] Henning Schulzrinne, Internet Media-on-Demand: The Real-Time Streaming Protocol

[MIRTSP] Leggio Simone, Streaming Media over the Internet with the Real Time Streaming Protocol

[BOL01] Raffaele Bolla, Servizi Multimediali e Qualità del Servizio (QdS) su IP. RTP – SDP

[BOL02] Raffaele Bolla, Servizi Multimediali e Qualità del Servizio (QdS) su IP. Session Initiation Protocol (SIP)

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

[SUN02] Sun Microsystem, Java Media Framework Api documentation