UNIVERSITÀ DEGLI STUDI DI TRENTO
Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in Informatica

Elaborato Finale

STRUMENTI PER L'ACQUISIZIONE DI VIDEO DI LEZIONI

Relatore: Dott. Marco Ronchetti

Laureando: Andrea Mittestainer

Anno Accademico

2006/2007

1. INTRODUZIONE

1.1 Premessa

Questa tesi nasce dalla necessità di riscrivere in Java alcune parti fondamentali del progetto L.O.D.E. a causa della scarsa diffusione, in ambiente universitario, del linguaggio Delphi utilizzato per la loro implementazione, cercando di non limitare le potenzialità e di non stravolgere la struttura originaria del progetto.

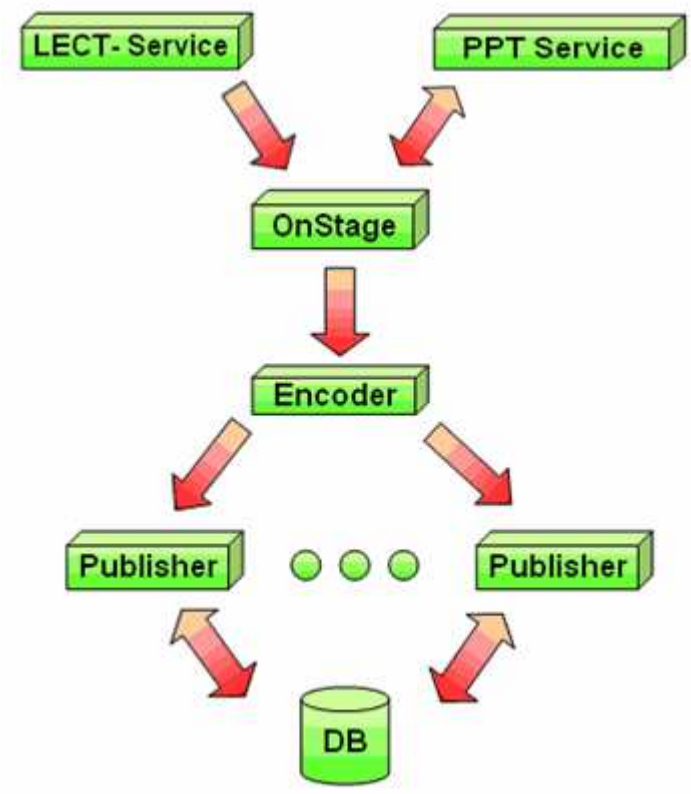
Il progetto L.O.D.E.[1] (Lectures On-Demand) è stato realizzato da Mauro Dolzani sotto la supervisione del docente Marco Ronchetti ed è un sistema per la produzione e la fruizione di lezioni multimediali.

Interfaccia L.o.d.e.

Dopo una breve esperienza con ePresence[2], un sistema sviluppato presso Knowledge Media Design Institute (KMDI) dell'Università di Toronto per permettere la divulgazione in rete di seminari e conferenze, a causa delle difficoltà incontrate nel trasporto dell'attrezzatura necessaria (2 pc, 1 videocamera e 1 set-top box), la presenza di un'interfaccia scarsamente user-friendly e uno scarso livello di automazione, è risultato necessario proporre una nuova architettura più snella, che potesse semplificare il lavoro dell'operatore e potesse abbassare i costi di gestione: L.O.D.E. nasce proprio con l'intento di trovare una soluzione a questi limiti.

L'immagine sottostante evidenzia la struttura di L.O.D.E per quanto riguarda la produzione delle lezioni. I primi due blocchi rappresentano il servizio di accesso ai dati che permette di caratterizzare la lezione che verrà ripresa definendone il titolo, il docente, la data, mentre l'altro box rappresenta

il processo di conversione dei contenuti della lezione che, ricevuta una presentazione in formato PowerPoint, restituisce un vettore di immagini contenente le singole slide.
Il blocco OnStage si occupa della registrazione video, della generazione dei riferimenti temporali ai contenuti utilizzati per permettere l'esatta ricostruzione della sequenza in fase di consultazione e dell'invio dei dati e del flusso video alla fase di PostProcessing, al termine della registrazione. In questa fase i flussi vengono ricodificati per generare gli streams che saranno poi effettivamente pubblicati e, terminata la codifica, il blocco Encoder ritrasmette la lezione e gli stream generati al servizio Publisher, che risiede sulla stessa macchina in cui è in esecuzione il server di streaming. Una volta ricevuti i dati questo blocco li salva sul database centrale per renderli disponibili alla consultazione. Essendo l'attività di streaming molto impegnativa è possibile suddividere il carico su più elaboratori avendo così un'istanza del servizio Publisher, che può ricevere lo stream, su ogni server di streaming.



Parte del sistema dedicata alla produzione della lezione

Le parti del progetto che saranno soggette a riscrittura in Java saranno quelle riguardanti l'acquisizione e la registrazione video, la conversione dei filmati in un formato più adatto alla distribuzione e la conversione delle presentazioni in singole immagini, affiancando all'utilizzo delle presentazioni in formato PowerPoint quelle in formato PDF.

2. TOOLS & API

Qui di seguito vengono descritti i tools e le api utilizzati nella riscrittura in Java dei processi di acquisizione, conversione video e conversione delle presentazioni PowerPoint e PDF in immagini.

2.1 Dsj - DirectShow Java wrapper

Dsj è un progetto che fornisce un java wrapper attorno alle Microsoft's DirectShow API[3] ed offre un insieme di classi di alto livello che permettono un facile accesso a Java a quelle funzionalità che i programmatori avevano abbandonato. Dal lato java Dsj tenta di essere il più trasparente possibile in modo da poter usare questo prodotto singolarmente o integrarlo ad esempio con le JMF API.
Dsj si appoggia ad una grande varietà di filtri DirectShow e in base a quelli installati sulla propria macchina è in grado di riprodurre e in molti casi codificare un discreto numero di formati:

Formati video	Formati audio
Wmv	mp3
DivX	wma
Asf	Vorbis (ogg)
Dv	wav
Theora (ogg)	aif
Mms & http streams	aac
Avi-synth	Mms & http radio
Mpeg 1 2 & 4	

Per tutti i formati video Dsj può inviare immagini non compresse a java e la maggior parte dei formati audio sono disponibili a java-sound postprocessing.
Oltre all'esecuzione delle operazioni di codifica e decodifica Dsj offre l'accesso alle API di acquisizione ed ad alcune strutture di alto livello di DirectShow che permettono il controllo di IEEE 1394 DV CamCoders e la riproduzione di DVD da Java. Infine dispone di una serie di funzionalità che permettono la costruzione ed il controllo di grafi DirectShow. Nella sua costruzione sono stati utilizzati altri progetti come:

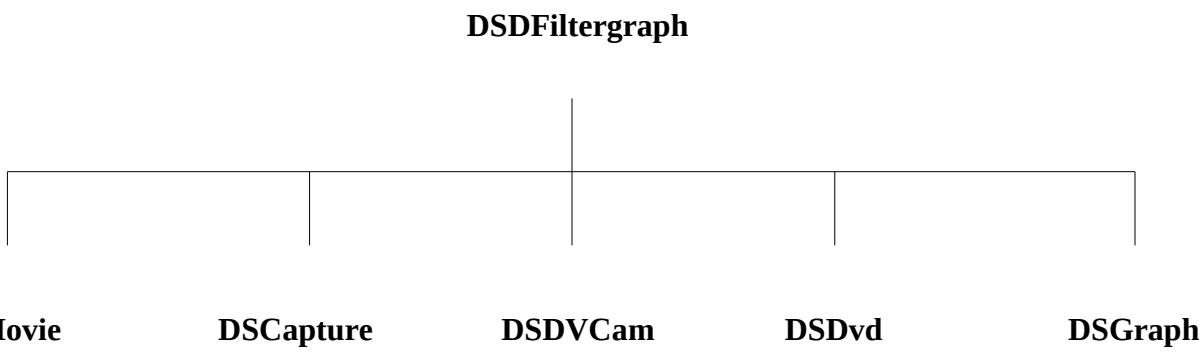
- **JMDS** - DirectShow Capture api Java wrapper:jmds.dev.java.net
- **fobs4jmf** - ffmpeg c++ & java bindings: http://fobs.sourceforge.net
- **java VLC** - VideoLan java bindings: http://jvlc.ihack.it
- **DXInput** - DirectInput Java wrapper:www.hardcode.de

- **jARToolkit** - ARToolkit java bindings: http://sourceforge.net/projects/jartoolkit/
- **jFFmpeg** - JMF codec pack: http://jffmpeg.sourceforge.net/

Il prodotto è in continuo sviluppo, non è da considerarsi terminato ed è consigliabile controllare il sito all'indirizzo http://www.humatic.de/htools/dsj.htm per eventuali aggiornamenti.

2.1.1 Struttura

La struttura del pacchetto non è molto articolata e consiste in una classe principale DSDFiltergraph, che con 5 sottoclassi racchiude le funzionalità base per le principali aree di utilizzo. Ci sono inoltre una serie di classi satelliti che forniscono i metodi di utilizzo generale e specifici per i vari ambiti.



Le classi contenute in DSDFiltergraph[4] sono:

- DSDMovie che permette la riproduzione di file o flussi multimediali e fornisce metodi per la codifica dei dati multimediali.
- DSDvd che mette a disposizione le funzionalità per l'utilizzo dei dvd.
- DSDVCam che contiene le funzionalità di DirectShow per IEEE1394 DV. Sono incluse le modalità framegrabbing e VCR includendo un controllo completo del dispositivo.
- DSCapture che permette l'accesso agli strumenti di cattura audio e video di DirectShow attraverso il modello WindowsDriverModel (standard secondo cui sono scritti la maggior parte dei driver per i sistemi operativi Microsoft). Lavora con la maggior parte di dispositivi di acquisizione, includendo schede TV e dispositivi mpeg2.
- DSGraph che riproduce un grafo DirectShow salvato in un file .grf o .xgr e che permette di costruirne uno ed eseguirlo mediante codice.

Le classi di supporto sono:

- DsEnvironment che gestisce le funzioni per il settaggio delle impostazioni generali;
- DsFilter referenzia un filtro DirectShow che è stato aggiunto ad un grafo;
- DsFilterInfo che contiene una struttura descrittiva attorno ai filtri DirectShow e non deve essere necessariamente in relazione con l'istanza inizializzata di un filtro descritto. DSFilterInfos sono costruiti mentre si interroga il dispositivo e vengono successivamente utilizzati quando i filtri vengono aggiunti al grafo.
- JavaSourceFilter permette di personalizzare gli oggetti DSFilter ed è in grado in inserire con Java informazione in un grafo;
- JavaSourceGraph è una sottoclasse di DSGraph e semplifica l'impostanzione del Java rendering all'interno del contesto DirectShow.
- SwingMovieController fornisce un controller per tutte le classi dsj.
- DSAudioStream classe annidata che contiene il flusso di byte rappresentante l'audio quando nel costruttore il bit è settato con il valore DELIVER_AUDIO.

Dalla versione 0.8 è stato introdotta DSGraph che offre la possibilità di creare grafi in Java collegando tra loro i filtri DirectShow necessari o eseguire file .grf o .xgr creati con Microsoft's GraphEdit.
Può essere utilizzata come Datasource per JMF, usata direttamente come player o in parallelo con QuickTime for Java. Inoltre può riprodurre i contenuti multimediali in finestre java standard o fullscreen usando DirectDraw o tecniche GDI con l'AWT o inviare il flusso video in un componente Swing autoridimensionabile.

Suffisso	**Sottoclasse generata**
.wmv .avi .mpeg ecc	DSMovie
.IFO .vob stringa nulla DVD dvd	DSDvd
.grf .xgr stringa xml	DSGraph
DV dv	DSDVCam

Corrispondenza tra valore passato come parametro nel costruttore e sottoclasse generata.

Rendering	Descrizione
Native (Direct Draw)	Fornisce alte prestazione e non permette l'accesso a Java ai dati delle immagini
Native (Native_Force_GDI)	Prestazione ancora più elevate rispetto al precedente, ma in alcune versione della jdk ci possono essere dei conflitti con Java2D. Java non può elaborare i bytes delle immagini.
Java side. Self redrawing (Java_AutoDraw)	Dsj esclude Java, elabora e trasemette i dati direttamente in un pannello della liberira Swing, che può essere direttamente aggiunto alla GUI.
Java side. Polled (Java_Poll)	Nessuna azione automatica. La ricezione ed elaborazione dei dati deve essere implementata via codice.
Java side. Polled & trasmissione dati RGB al posto di BGR (Java_Poll_Rgb)	Utilizzata per adattare JMF alla trasmissione RTP.
Headless	Non interviene con il flusso di dati DirectShow
Deliver_Audio	Ottiene il flusso audio

Tipi di rendering.

2.1.2 Installazione

Per poter utilizzare questo prodotto bisogna:

- installare DirectX9 e Windows Media Player 9 o superiori;
- posizionare il file “dsj.dll” nella cartella System32 o nella directory della jre;
- posizionare il file “dsj.jar” in jre/lib/ext o includerlo nel progetto che si sta creando.

2.2 Ffmpeg

FFmpeg[5] è un convertitore audio e video molto veloce che può anche estrarre un flusso audio/video da una sorgente in realtime. L'interfaccia a linea di comando è progettata per essere intuitiva, nel senso che questo programma tenta di impostare in maniera autonoma la maggior parte dei parametri. L’unica variabile che bisogna specificare è il bitrate desiderato.

2.2.1 Installazione

L’installazione sotto Windows comporta l’utilizzo di alcuni strumenti aggiuntivi come MinGW, MSYS e WINCVS[6] e i passi da seguire sono:

- Scaricare la versione corrente di MinGW e MSYS (Window Exe Binaries MSYS-1.0.10.exe & MinGW-3.1.0-1.exe);
- Installare MinGW;
- Installare MSYS. Nella fase di post installazione sarà richiesto di inserire il path di MinGW che andrà inserito preceduto da "/". Nel caso si immettesse in modo errato ripetere senza alcun timore la procedura di installazione di MSYS.
- Scaricare LAME (mp3 codec open-source) e scompattarlo nella cartella di MSYS;
- Aprire MSYS usando il file "msys.bat" e dopo essersi spostati nella cartella di Lame digitare i seguenti comandi:
  - ./configure;
  - make;
  - make install.
- Scaricare WINCVS (http://www.wincvs.org) ed installarlo. Aprire la finestra Command-Line premendo CTRL-L e digitare il comando:
  - *cvs -z9 -d:pserver:anonymous@mplayerhq.hu:/cvsroot/ffmpeg co ffmpeg*
- Selezionare l'opzione "*Execute for directory*" e selezionare la directory di MSYS dove verranno salvati i file scaricati. Premere INVIO e l'ultima versione di FFmpeg sarà scaricata sul computer;
- Aprire MSYS sempre con il file "msys.bat" spostarsi nella directory dove si trovano i file appena scaricati e digitare:
  - *./configure --enable-memalign-hack --enable-mingw32 --enable-mp3lame --extra-cflags=-I/local/include --extra-ldflags=-L/local/lib;*
  - *Make;*
  - *Make install (*opzionale*).*

In alternativa se non si vuole seguire questa procedura è possibile scaricare direttamente l'eseguibile da http://www.videohelp.com/tools?tool=263.

2.2.2 Estrazione Video ed Audio

FFmpeg può usare una sorgente compatibile con video4linux e qualsiasi sorgente audio OSS (Open Sound System) e la sintassi da utilizzare è:

ffmpeg /tmp/out.mpg

Notare che è necessario attivare il canale e la sorgente video corretti prima di avviare ffmpeg con un qualsiasi visualizzatore TV, come l'xawtv di Gerd Knorr. Bisogna anche impostare correttamente il livello di registrazione audio con l'ausilio di un normale mixer.

2.2.3 Conversione di file Video ed Audio

FFmpeg come input può utilizzare qualsiasi formato o protocollo. Le principali operazioni di conversione che si possono eseguire sono le seguenti:

- Passare da un tipo di file ad un altro;
- Unire tra loro una sequenza di file YUV;
- Impostare diversi file di ingresso e di uscita;
- Eseguire una conversione audio e video nello stesso tempo;
- Codificare diversi formati nello stesso tempo e definire un "mappa degli indici" dal flusso di ingresso a quelli di uscita;
- Decodificare i VOB decriptati.

2.2.4 Sintassi

La sintassi generica è: *ffmpeg [[options][-i ingresso_file]]... {[options] uscita_file}...* Se nessun file in ingresso viene fornito, viene eseguita la cattura audio/video. Come regola generale, le opzioni vengono applicate al file che viene successivamente specificato. Ad esempio, fornendo l'opzione '-b 64', viene impostato il bitrate relativo al prossimo file. Potrebbe essere necessario fornire l'opzione sul tipo di formato quando vengono usati come ingresso i file privi di header (RAW file). Normalmente Ffmpeg prova a convertire con la minor perdita possibile di qualità: per i file di uscita viene usato lo stesso parametro riferito all’audio ed al video specificato per i file in ingresso.

2.2.5 Opzioni principali

- '-L' mostra licenza
- '-h' mostra help
- '-formats' mostra i formati, codec, protocolli disponibili.
- '-f fmt' forza il formato
- '-i filename' nome file in ingresso
- '-y' sovrascrivi il file in uscita
- '-t duration' imposta (in secondi) il tempo di registrazione. La sintassi "hh:mm:ss[.xxx]" è supportata.
- '-title string' imposta il titolo
- '-author string' imposta l'autore
- '-copyright string' imposta i diritti d'autore
- '-comment string' imposta un commento
- '-b bitrate' imposta il bitrate video (in kbit/s).

2.2.6 Opzioni video

- '-s size' imposta la dimensione dei frame [160x128]
- '-r fps' imposta il frame rate [25]
- '-b bitrate' imposta il bitrate video (in kbit/s) [200]
- '-vn' disabilita la registrazioni video [no]
- '-bt tolerance' imposta la tolleranza di bitrate per il video (in kbit/s)
- '-sameq' usa la stessa qualità video della sorgente (implica VBR)
- '-pass n' seleziona il numero di passaggi (1 o 2) da eseguire nella codifica. E' quindi necessario per effettuare la codifica a due passaggi (two pass encoding). Le statistiche del video (log) vengono registrare nel primo passaggio ed il video invece viene generato, all’esatto bitrate richiesto, durante il secondo passaggio.
- -passlogfile file' seleziona il nome del file log per la codifica a due passaggi

2.2.7 Opzioni audio

'-ar freq' imposta la frequenza di campionamento audio [44100]

'-ab bitrate' imposta il bitrate del flusso audio in kbit/s [64]

'-ac channels' imposta il numero di canali audio [1]

'-an' disabilita la registrazione audio [no]

2.3 Graphedit

GraphEdit è un'applicazione con interfaccia grafica che permette di costruire grafi per combinare insieme, in modo sequenziale, l'azione dei filtri DirectShow[7] installati sul computer, cambiandone anche, in alcuni casi, i parametri di funzionamento. Permette di testare i grafi prima di inserirli nell'applicazione che si andrà a scrivere, inserire filtri personalizzati in un grafo e verificare il corretto funzionamento.

Ogni filtro viene rappresentato con un rettangolo, con ai lati dei piccoli rettangolini rappresentanti i pins. I pins di input si trovano sulla sinistra mentre quelli di output sulla destra. Le frecce identificano la connessione tra i vari filtri.

Con GraphEdit si è in grado di:

- Creare o modificare grafi usando un'interfaccia che permette di rimuovere ed aggiungere i filtri facilmente;
- Simulare la chiamata per la costruzione di un grafo;
- Avviare, stoppare, mettere in pausa un grafo;
- Controllare quali filtri sono installati sulla macchina e le loro proprietà;
- Aggiungere nuovi filtri;
- Vedere i tipi multimediali di ogni pin.

2.3.1 Installazione Programma

GraphEdit è compreso nel pacchetto *Microsoft DirectX SDK* e il metodo più comune per ottenerlo è quello di scaricare tutto il pacchetto di sviluppo delle DirectX. Se invece si vuole scaricare solo il programma si può comunque trovarlo facilmente nel web, partendo da Google con una semplice ricerca. Una volta scaricato, il programma non necessita di alcuna installazione e può essere subito utilizzato.

2.3.2 Costruzione Grafo Acquisizione Video

Prima di procedere alla creazione del grafo di acquisizione bisogna installare sulla macchina il codec XviD. Per fare questo dobbiamo scaricare K-Lite Codec Pack Full. Dopo averlo installato, partendo da Start proseguire in Programmi → K-Lite Codec Pack → Configuration → XviD encoding. A questo punto cliccare su "Other Options" e deselezionare la cella "Display Encoding Status". Salvare le modifiche ed uscire.

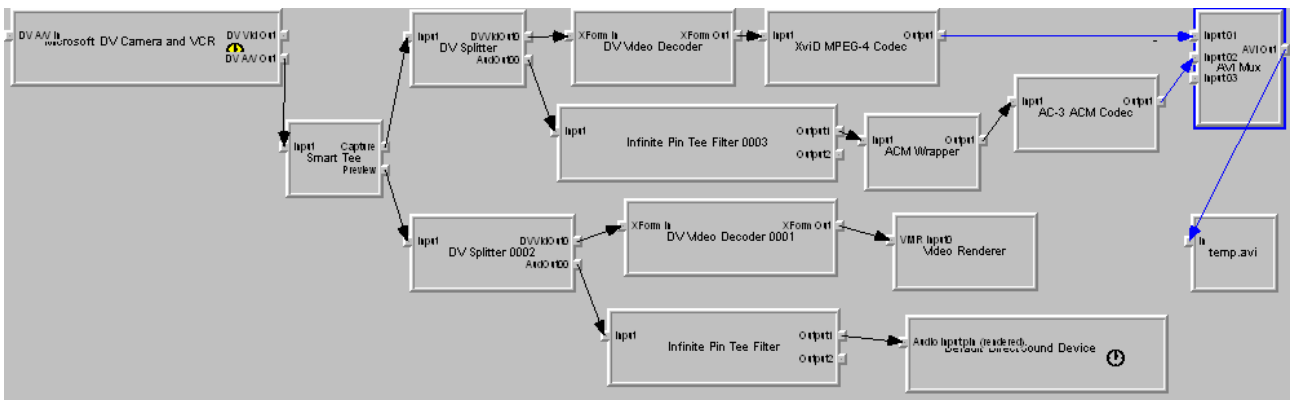
Nel caso in cui venisse utilizzata esclusivamente una videocamera digitale per acquisire sia il flusso video che audio i passi per la costruzione del grafo sono i seguenti:

- Collegare la videocamera al Computer;
- Lanciare GraphEdit.exe;
- Inserire i seguenti filtri:
    1. (categoria Video Capture Sources) Microsoft DV Camera and VCR;
    2. (categoria DirectShow Filters) Smart Tee;
    3. (categoria DirectShow Filters) DV Splitter;
    4. (categoria DirectShow Filters) DV Video Decoder;
    5. (categoria Video Compressor) XviD MPEG-4 Codec;
    6. (categoria DirectShow Filters) Avi Mux;
    7. (categoria DirectShow Filters) File Writer ed inserire il nome "temp.avi" nella cartella desiderata;
    8. (categoria DirectShow Filters) DV Splitter;
    9. (categoria DirectShow Filters) Infinite Pin Tee Filter;
    10. (categoria DirectShow Filters) Infinite Pin Tee Filter;
    11. (categoria Audio Renderers) Default DirectSound Device;
    12. (categoria DirectShow Filters) ACM Wrapper;
    13. (categoria Audio Compressor) AC-3 ACM Codec;
    14. (categoria DirectShow Filters) DV Video Decoder;

15. (categoria DirectShow Filters) Video Renderer;

- Collegare i seguenti nodi:
  1. Microsoft DV Camera and VCR DV A/V Out con Smart Tee Input;
  2. Smar Tee Capture con DV Splitter Input;
  3. DV Splitter DVVidOut0 con DV Video Decoder XForm In;
  4. DV Video Decoder XForm Out con XviD MPEG-4 Codec Input;
  5. XviD MPEG-4 Codec Output con AVI Mux Input 01;
  6. AVI Mux AVI Out con temp.avi In;
  7. DV Splitter AudOut00 con Infinite Pin Tee Filter Input;
  8. Infinite Pin Tee Filter Output1 con ACM Wrapper Input;
  9. ACM Wrapper Output con AC-3 ACM Codec Input;
  10. AC-3 ACM Codec Output con AVI Mux Input02;
  11. Smart Tee Preview con DV Splitter Input;
  12. DV Splitter DVVidOut0 con DV Video Decoder XForm In;
  13. DV Video Decoder XForm Out Video Render Input.
  14. DV Splitter1 AudOut00 con Infinite Pin Tee Filter Input;
  15. Infinite Pin Tee Filter Output con Default DirectSound Device Audio Input pin.
- Premere il tasto destro del mouse sul filtro DV Video Decoder, cliccare Filter Propreties e selezionare la risoluzione di decodifica Dimezzata (NTSC: 360*240, PAL: 360*288). Premere OK;
- Salvare il grafo.

A questo punto il grafo è completo e per verificare che funzioni correttamente può essere eseguito premendo il bottone Run.



Il primo filtro identifica la sorgente, ovvero la videocamera connessa al pc mediante la presa DV, alla quale viene collegato il box “Smart Tee” che permette di mantenere distinte le operazioni di cattura e preview del filmato. Per poter salvare il filmato è necessario codificare i flussi audio e video separatamente e l'operazione svolta dal filtro “DV Splitter” è proprio quella di isolarli per una loro successiva elaborazione. L'audio mediante la sequenza di filtri “Infinite Pin Tee Filter” → ”ACM Wrapper” →”AC-3 ACM Codec” viene codificato nel formato AC-3, mentre il video con la sequenza “DV Video Decoder” → ”XviD MPEG-4 Codec” viene codificato nel formato mp4. A questo punto i flussi possono essere uniti mediante il filtro “AVI Mux” e scritti su file con il box “File writer”.

Per affrontare il processo che genera la preview durante la ripresa è necessario ritornare al filtro “Smart Tee”. A questo è stato connesso un filtro “DV Splitter”, con lo scopo di gestire separatamente, anche in questo caso, il flusso audio e vivo. Per il video viene connesso al filtro che sdoppia i flussi il box “DV Video Decoder”, che decodifica un flusso DV in un flusso video non compresso e successivamente il filtro “Video Renderer” che visualizza quanto viene ripreso. Per la riproduzione dell’audio è connesso al “DV Splitter” il box “Infinite Pin Tee Filter” e successivamente il filtro “Default DirectSound Device”.

Le combinazioni di apparecchi per l’acquisizione audio e video sono molteplici e non si è affatto vincolati ad utilizzare la strumentazione a cui fa riferimento il grafo appena mostrato. Se ad esempio avessimo la possibilità di utilizzare un radio-microfono collegato ad una scheda sonora basterebbe apportare alcune semplici modifiche al grafo precedente.

La prima passo da compiere è aggiungere il filtro della scheda sonora, selezionandolo dalla categoria Audio Capture Sources, successivamente, scorrendo il ramo Capture che parte dal filtro Smart Tee, eliminare la relazione tra il filtro Dv Splitter e Infinite Pin Tee Filter ed infine collegare il filtro della scheda sonora con il pin in entrata del filtro Infinite Pin Tee Filter.

2.4 JPedal

JPedal (Java Pdf Extraction Decoding Access Library)[8] è una libreria scritta completamente in Java che permette la gestione di file Pdf. Le sue principali caratteristiche sono:

- prevede il supporto di una grande varietà di font: TrueType, Type 0, 1, 1C e 3, embedded, subsetted and CID, Identity H&V permettendo così di visualizzare correttamente tutti i file pdf creati con questo standard.
- Permette l'estrazione di porzioni o dell'intero testo da singole pagine, interi documenti e tabelle.

- Consente la visualizzazione, elaborazione, stampa ed estrazione dei contenuti da FDF forms, permette di aggiungere a quest'ultime degli ascoltatori e di usare strumenti standard o personalizzati per visualizzare e salvare forms.
- Supporta una vasta gamma di color space: DeviceRGB, CalRGB, DeviceGRAY, CalGRAY, ICC, indexed, DeviceCMYK e DeviceN. Jpedal è in grado di leggere e visualizzare i formati jpg, tiff, gif.
- Permette l'estrazione di immagini in blocco o singole presenti in un documento salvandole in diversi formati.
- Consente di salvare un documento in singole immagini in modo automatico, permettendo di scegliere il formato di output (png, tif, jpg).

2.4.1 Installazione

Per poter utilizzare questa libreria visitare la pagina a questo indirizzo http://www.jpedal.org/download.php, scaricare l'ultima versione del prodotto, il pacchetto contenente i file aggiuntivi necessari per far funzionare la liberia correttamente e posizionare il tutto nella cartella della Java Virtual Machine sotto jre/lib/ext.

2.5 Jacob

Jacob è un Java-Com Bridge che permette di chiamare componenti COM Automation da Java[9]. Utilizza le JNI per effettuare chiamate native a COM ed alle librerie WIN32. La sola documentazione presente è quella JavaDoc e per poter prendere confidenza con questo tool bisogna basarsi su esempi reperiti dalla rete o forum dedicati.
Microsoft Component Object Model (COM) [16] è una tecnologia che permette la comunicazione tra i vari componenti software all'interno dei sistemi operativi. È utilizzata dai programmatori per creare componenti software riusabili, collegare componenti tra loro per costruire le applicazioni ed ottenere vantaggi dai servizi di Windows.

2.5.1 Installazione

Per installare questo pacchetto visitare il sito al link http://sourceforge.net/project/showfiles.php?group_id=109543, scaricare l'ultima versione del progetto e posizionare il file .jar all'interno della cartella della Java Virtual Machine in jre/lib/ext e il file .dll nella cartella System32.

2.6 Jmf 2.0

Java Media Framework sono API[10] create per permettere di incorporare tipi di dati multimediali in applicazioni o applet Java come file con estensione: MPEG-1, MPEG-2, Quick Time, AVI, WAV, AU e MIDI.

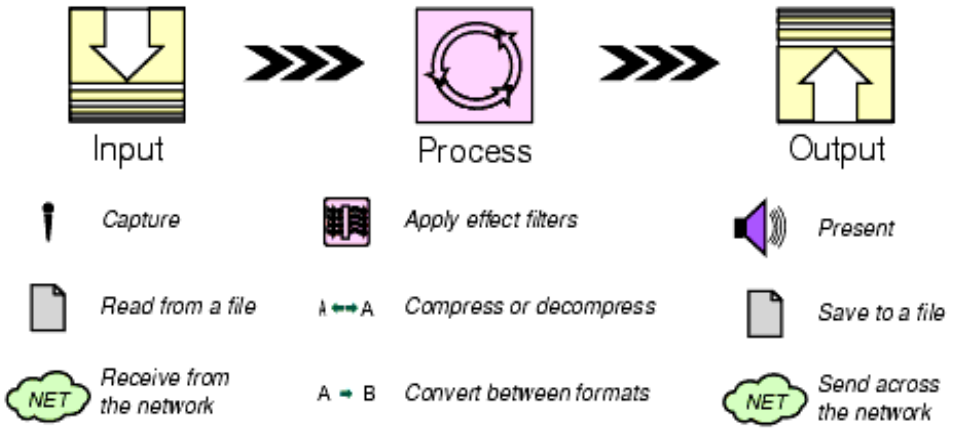
Usando JMF si ha la possibilità di riprodurre gli standard sopra citati e di gestirli su base temporale. Com'è ben noto, la peculiarità di Java è l'utilizzo di una Java Virtual Machine (JVM) che, tramite emulazione software (utilizzando esclusivamente la CPU), interpreta il byte-code generato dal compilatore Java. Questo meccanismo permette la portabilità del codice su più piattaforme ma, nel caso in cui sia richiesta una elevata velocità di esecuzione, impone dei seri vincoli di prestazioni. Il trattamento di dati Multimediali è uno dei casi in cui è richiesta un'elevata velocità computazionale (decompressione delle immagini, rendering, ecc.) e non è quindi sufficiente la sola emulazione della CPU per ottenere delle prestazioni ottimali. Ogni sviluppatore che desideri implementare un lettore multimediale in Java, e voglia ottenere delle prestazioni eccellenti deve, necessariamente, ricorrere al codice nativo della piattaforma alla quale è interessato incontrando così due problemi:

- Specifica conoscenza delle funzioni native da parte del programmatore.
- Un programma Java che utilizza codice nativo non è più trasportabile su piattaforme diverse da quella originaria.

Le API JMF tentano di risolvere questi problemi, mettendo a disposizione del programmatore una serie di chiamate ad "alto livello" per la gestione del codice nativo e l'applicazione o l’applet non necessitano di conoscere quando e se deve sfruttare particolari metodi nativi per svolgere una determinata azione.

Con il passaggio dalla versione 1.0 alla 2.0 è stato estesa la struttura per permettere la cattura e l’archiviazione dei dati multimediali, il controllo dei tipi dei processi che vengono eseguiti durante il playback. Sono stati inoltre definiti dei plug-in che permettono di personalizzare ed estendere le funzionalità delle JMF ed è stato incluso nelle API il MediaPlayer Java Bean che può essere istanziato direttamente e può rappresentare uno o più flussi multimediali.

File audio, video, animazioni, MIDI posso essere definiti time-based media in quanto cambiano significativamente rispetto al tempo. Possono essere ottenuti con diversi dispositivi come microfoni, web-cam, videocamere, file locali o remoti.

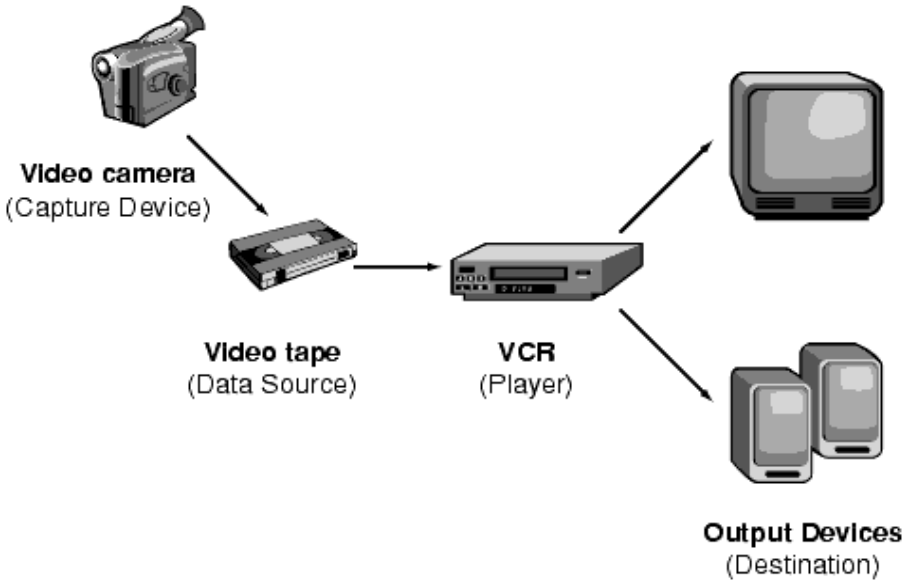


Media processing model

Sfruttando i vantaggi delle piattaforma JAVA, JMF risolve il problema di "scrivere una volta ed eseguire ovunque" aumentando così la capacità dei sistemi operativi e permettendo agli sviluppatori di creare facilmente programmi Java con alta portabilità che trattano dati multimediali.

2.6.1 Struttura Jmf

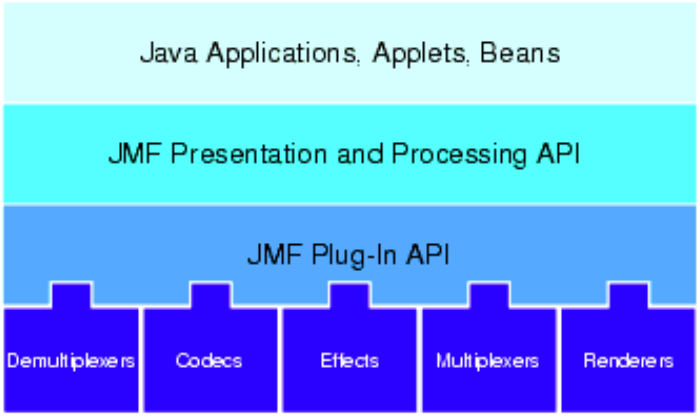
Struttura ad alto livello



Jmf usa lo stesso procedimento per la registrazione, elaborazione e rappresentazione dei filmati rappresentato in figura. Un data source raccoglie i flussi multimediali come una videocassetta e il player esegue procedure di elaborazione e controllo come un VCR. Per effettuare le operazioni di acquisizione e presentazione JMF necessita di particolari strumenti come microfoni, web-cam, videocamere, monitor, altoparlanti.

DataSource e Players sono parti integrante delle JMF API di alto livello per gestire la cattura, la rappresentazione e l'elaborazione di dati multimediali. JMF dispongono inoltre di una serie di API di basso livello che supportano senza giunture l'integrazione di componenti personalizzati ed estensioni.

Questa stratificazione permette agli sviluppatori un facile utilizzo di queste API per inserire dati multimediali nei loro programmi mantenendo la flessibilità e l'estensibilità richiesta per supportare avanzate applicazioni e future tecnologie multimediali.



Architettura ad alto livello

Managers

JMF API sono costituite principalmente da interfacce che definiscono i comportamenti e le interazioni degli oggetti usati per acquisire, elaborare e rappresentare dati multimediali. Usando oggetti intermedi chiamati managers JMF rende facile l'integrazione di nuove implementazioni delle interfacce chiave, che possono essere usate senza problemi con le classi esistenti.

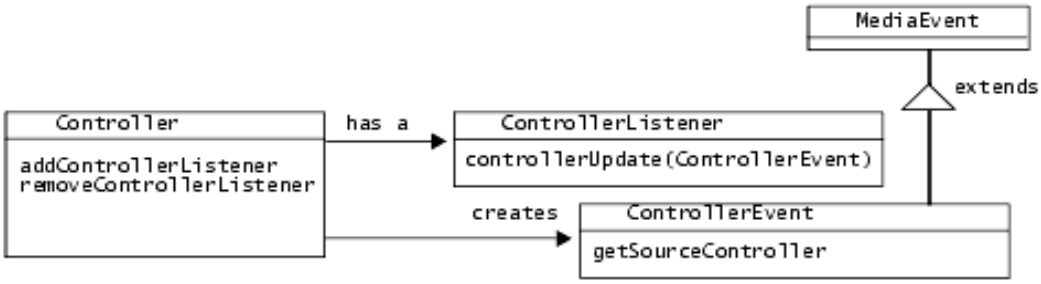
Vengono usati quattro tipi di Managers:

- Manager: permettono la costruzione di Players, Processors, DataSources, e DataSinks. Dal lato client questi oggetti vengono costruiti sempre nello stesso modo. L'oggetto richiesto viene costruito partendo da un'implementazione già definita o da un'implementazione personalizzata;
- PackageManager: mantiene un registro dei packages che contengono le classi della JMF, come Players, Processors, DataSources e DataSinks personalizzati;
- CaptureDeviceManager: contiene un registro dei dispositivi di cattura disponibili;
- PlugInManager: contiene il registro dei plug-in per l'elaborazione dei dati disponibili, come Multiplexers, Demultiplexers, Codecs, Effects, e Renderers.

Per scrivere un programma che faccia uso della JMF abbiamo bisogno dei Manager per costruire Players, Processors, DataSources e DataSinks. Se vogliamo catturare dei dati da un dispositivo di input, come ad esempio una videocamera dovremmo usare CaptureDeviceManager per trovare quali dispositivi sono disponibili e per ottenere informazioni da essi; se siamo interessati a sapere che operazioni può svolgere un processore sui dati dobbiamo eseguire una query sul PlugInManager per determinare quali plugin sono registrati; se vogliamo implementare un nuovo plug-in possiamo registrarlo con il PlugInManager per permettere al Processors di accedervi. Per utilizzare un Player, Processor, DataSource o DataSink modificati dobbiamo utlizzare PackageManager per registrare l'indirizzo univoco del package contenente gli oggetti manipolati.

Event Model

JMF usa un meccanismo a eventi per tenere i programmi, che utilizzano questa libreria, informati sul corrente stato del media system e per abilitarli a rispondere ai possibili errori. Ogni volta che un oggetto JMF ha bisogno di riferire sul suo stato posta un MediaEvent, che ha il compito di identificare alcuni tipi particolari di eventi e segue il modello di eventi stabilito da Java Beans.



Per ogni tipo di oggetto che richiede un MediaEvents, JMF definisce una corrispondente interfaccia e per ricevere la notifica che un MediaEvent è stato postato, bisogna implementare la corretta interfaccia e registrare la classe del listener con l'oggetto che posta l'evento chiamando il suo addListener .

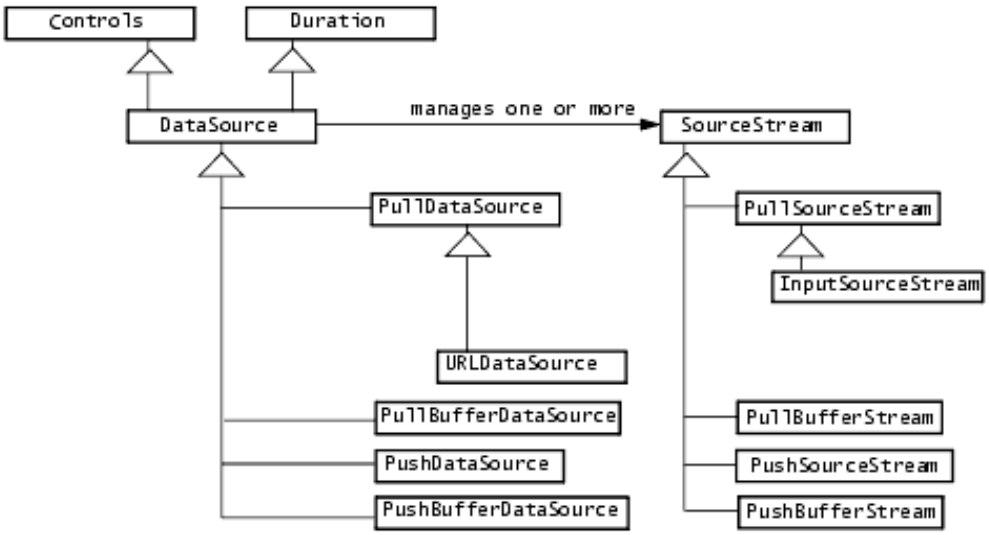
Oggetti come Player e Processor e alcuni oggetti di controllo come GainControl postano MediaEvents.

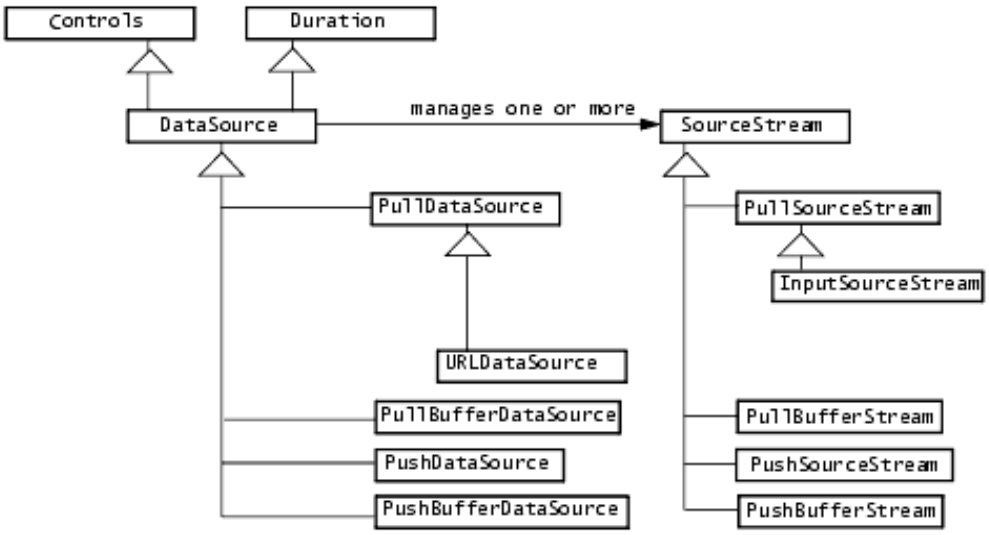
Data Model

Il media player di JMF usa i DataSource per gestire il trasferimento di contenuti multimediali che contengono la posizione del file multimediale, il protocollo e il software utilizzato per trasmettere il contenuto. Un DataSource può essere identificato sia da un JMF MediaLocator che da un URL. I

MediaLocators sono simili agli URL e possono essere costruiti partendo da un URL, ma solo se il corrispondente protocollo di gestione non è installato sul sistema. Un DataSource gestisce una serie di SourceStream, e normalmente usa un array di byte come unità di trasferimento. Nel caso in cui avessimo un buffer come data source useremo un Buffer di oggetti come unità di trasferimento.
JMF definisce 2 tipi di oggetto DataSource:

- Push Data Sources;
- Pull Data Sources.





Push and Pull Data Sources

I dati multimediali possono esser ottenuti da diversi tipi di risorse e i JMF data source possono essere classificati in base a quando il trasferimento dei dati ha inizio:

- Pull Data-Source: il client da inizio al trasferimento dei dati e controlla il flusso dal pull data-sources; il protocollo usato per il trasferimento è http o file. JMF definisce due tipi di

pull data-source: PullDataSource e PullBufferDataSource che usa un Buffer di oggetti come unità di trasferimento;

- Push Data-Source: il server inizia il trasferimento e controlla il flusso di dati da un push data source, che possono essere broadcast media, multicast media e video-on-demand (VOD). I protocolli di trasporto utilizzati sono ad esempio per il primo il Real-time Transport Protocol (RTP) e il MediaBase per il VOD. JMF definisce due tipi di push data-source: PushDataSource e PushBufferDataSource.

Il grado di controllo che un utente ha sui dati multimedia dipende dal tipo di dato che si sta utilizzando, ad esempio se abbiamo un file MPEG l'utente è abilitato ad utilizzare la funzione di replay o a spostarsi in un punto qualsiasi del filmato. Il broadcast media, invece, è gestito da un server e non è possibile spostarsi in un punto desiderato del filmato. Alcuni protocolli VOD implementano controlli limitati da parte dell' utente, come per esempio i programmi che permettono all'utente di posizionarsi dove vuole nel filmato, ma non gli permettono il forward e il rewind.

DataSource speciali

In queste API sono definiti 2 tipi speciali di data source, cloneable data sources e merging data sources. Il primo è usato per creare cloni di pull e push DataSource e per effettuare questa operazione chiamiamo dal Manager il metodo createCloneableDataSource e gli passiamo il DataSource che vogliamo clonare.

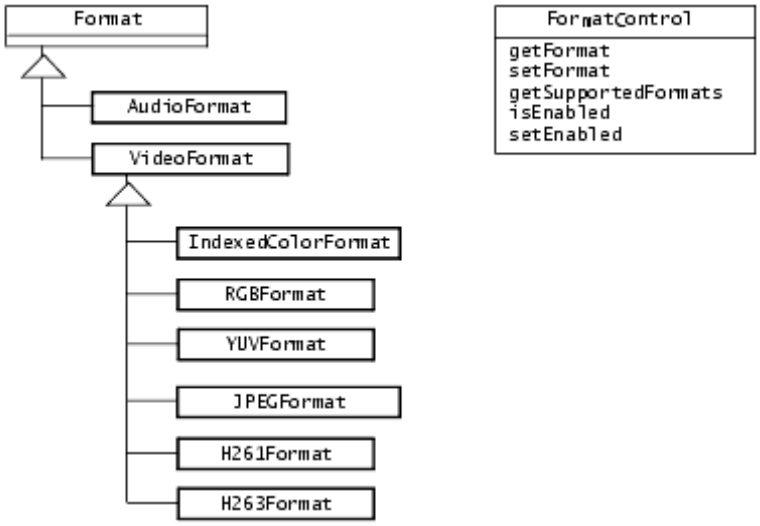
Quando un DataSource è stato passato al metodo createCloneableDataSource, dovremmo interagire solamente con il DataSource clonabile e i suoi cloni, e non usare più direttamente il DataSource originale.

Cloneable data sources implementa l'interfaccia SourceCloneable, che contiene il solo metodo createClone. Chiamandolo possiamo creare un qualsiasi numero di cloni di un DataSource che è stato usato per costruire il DataSource clonabile. I cloni possono essere controllati attraverso il DataSource clonabile ed infatti quando vengono chiamati i metodi di connessione, disconnessione, avvio e stop su quest'ultimo vengono propagati a tutti i suoi cloni. I cloni non necessariamente devono avere le stesse caratteristiche del data source clonabile usato per crearli o del DataSource originale, per esempio un data source creato per un dispositivo di acquisizione dati deve funzionare come master data source per i suoi cloni perchè finché non si smette di usarlo i suoi cloni non svolgeranno alcuna operazione. Se agganciamo il data source clonabile ad uno o più cloni, quest'ultimi produrranno dati contemporaneamente al master.

Un MergingDataSource è usato per unire il SourceStream proveniente da diversi DataSource in un unico DataSource. Questo permette la gestione centralizzata dei DataSource perché i metodi chiamati sul MergingDataSource vengono propagati ai DataSource uniti.
Per costruirne uno dobbiamo chiamare il metodo createMergingDataSource sempre attraverso il Manager e passargli un array che contiene tutti i DataSource che vogliamo unire. I DataSource devono essere tutti dello stesso tipo, o PullDataSource o PushDataSource e il tipo del contenuto sarà application/mixed-media.

Data Formats

Il formato multimediale di un oggetto è rappresentato dall'oggetto Format, che non contiene i parametri di codifica specifici o informazioni sulla distribuzione temporale, ma informazioni riguardanti il nome del formato da codificare e il tipo di dato richiesto.



Esaminiamo alcune caratteristiche di questo oggetto:

- AudioFormat descrive gli attributi specifici di un formato audio quali rate, bits, numero di canali.
- ViedoFormat invece contiene informazioni relative ai dati video. I più comuni formati video sono:
    - IndexedColorFormat ;
    - RGBFormat ;
    - YUVFormat;
    - JPEGFormat;
    - H261Format;

- H263Format.

Per ricevere notifica di modifiche di un formato da un Controller, bisogna implementare l'interfaccia ControllerListener e ascoltare il FormatChangeEvents.

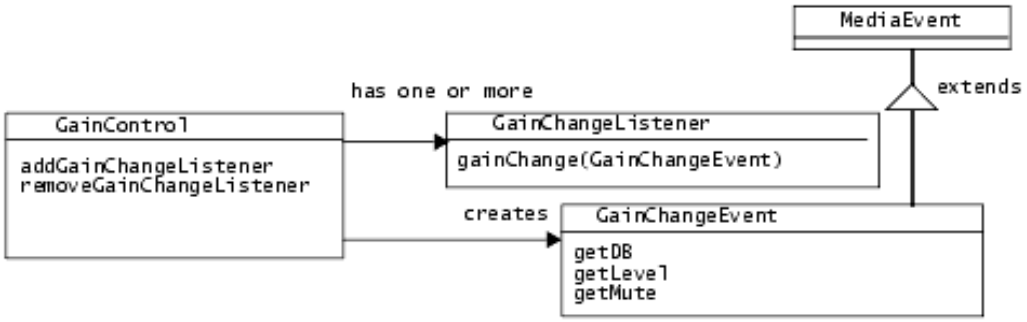
Control

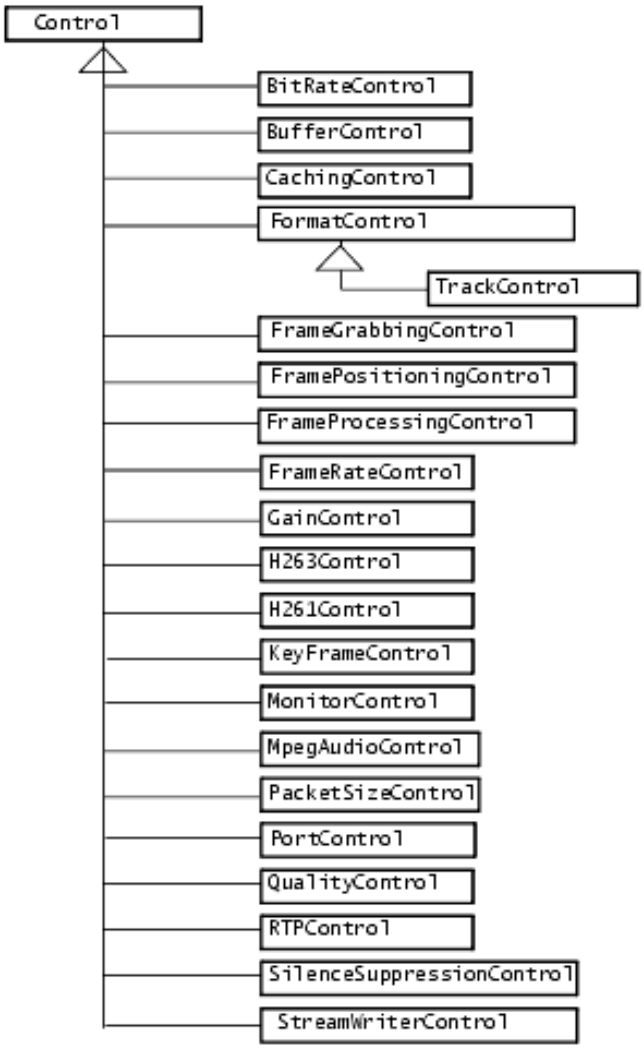
JMF Control fornisce meccanismi per interrogare e settare gli attributi degli oggetti. Un Control spesso permette l'accesso ad un componente di un'interfaccia utente che abilita l'utente al controllo degli attributi di un oggetto.
Qualsiasi oggetto JMF che vuole permettere l'accesso al suo corrispondente oggetto Control deve implementare l'interfaccia Control che definisce i metodi per ricevere gli oggetti Control associati. DataSource e PlugIn usano questa interfaccia per permettere l'accesso ai loro oggetti Control.

Standard Controls

Le API definiscono l'iterfaccia Control in figura sotto. Esaminiamo qualche metodo di questa interfaccia:

- CachingControl: permette di monitorare e visualizzare il progresso di un download. Player o Processor, che visualizzano barre di progresso di un donwnload, implementano questa interfaccia.
- GainControl: permette di modificare l'audio di un file, come il livello o il muto, in un Player o Processor. Supporta inoltre un meccanismo di listener per la modifica del volume.





- StreamWriterControl: oggetti come il DataSink e il Multiplexer che leggono dati da un DataSource e li scrivono, ad esempio in un file, implementano questa interfaccia, questo Control permette di limitare la dimensione dello stream che sta creando;
- FramePositioningControl e FrameGrabbingControl: esporta le operazioni da effettuare sui frame per i Player e i Processor. Il primo permette il posizionamento preciso in un determinato frame con un oggetto di tipo Player o Processor. Il secondo fornisce meccanismi per il grabbin di un frame da un flusso video. FrameGrabbingControl è anche supportato al livello Render;
- FormatControl: gli oggetti che hanno un formato devono implementare l'interfaccia FormatControl per permettere l'accesso al Format e poter effettuare query e cambiamenti al formato. TrackControl è un tipo di FormatControl che permette di controllare quali azioni saranno svolte da un Processor su una traccia multimediale. Con questo metodo possiamo

specificare quale formato di conversione sarà applicato sulle varie tracce e selezionare Effect, Codec, e Renderer plug-ins che sono usati dal Processor;

- PortControl: definisce i metodi per controllare l'output di un dispositivo di acquisizione;
- MonitorControl: permette una prima visione dei dati così come sono stati acquisiti o decodificati;
- BufferControl: abilita il livello utente a controllare le operazioni di buffering di un determinato oggetto.

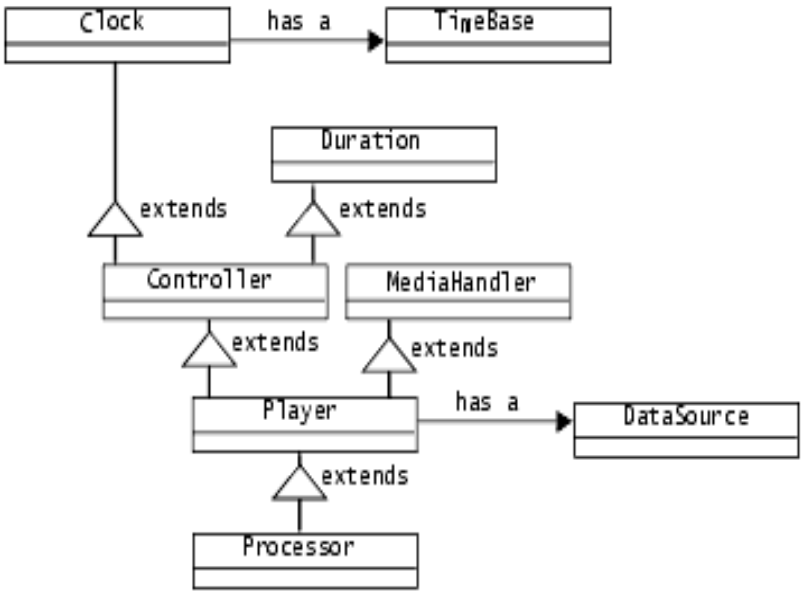
JMF definisce inoltre una serie di codec di controllo che permettono di modificare alcuni parametri dei contenuti multimediali scavalcando codificatori e decodificatori hardaware o software. Questi sono BitRateControl, FrameProcessingControl, FrameRateControl, KeyFrameControl, MpegAudioControl, QualityControl, SilenceSuppressionControl.

Componenti interfaccia utente

I Control permettono di accedere all'interfaccia utente Component che mostra i suoi controlli di funzionamento all'utente finale. Per avere l'interfaccia standard per un particolare Control, bisogna chiamare getControlComponent, ritornando così un componente AWT che può essere aggiunto ad un applet o ad una finestra di un'applicazione.
I Controller permettono inoltre l'accesso all'interfaccia Components. Se non si vuole usare i componenti di controllo base, questi possono essere implementanti a nostro piacimento ed usare gli event listener per determinare quando devono essere eseguiti.

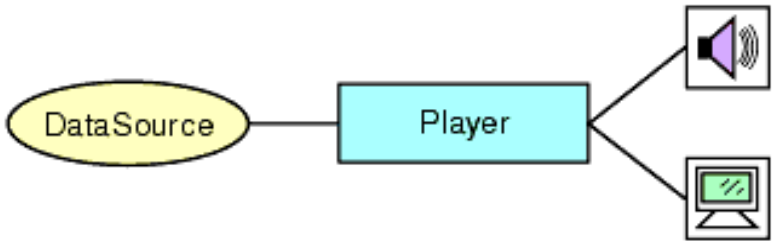
Presentazione

I processi di presentazione sono regolati dall'interfaccia Controller, che definisce gli stati e i meccanismi di controllo per un oggetto che controlla, presenta o acquisisce un dato multimediale.
Le JMF API implementano due tipi di Controllers: Player e Processor che sono costruiti per un particolare data source e normalmente non sono riutilizzati per presentare altri dati multimediali.



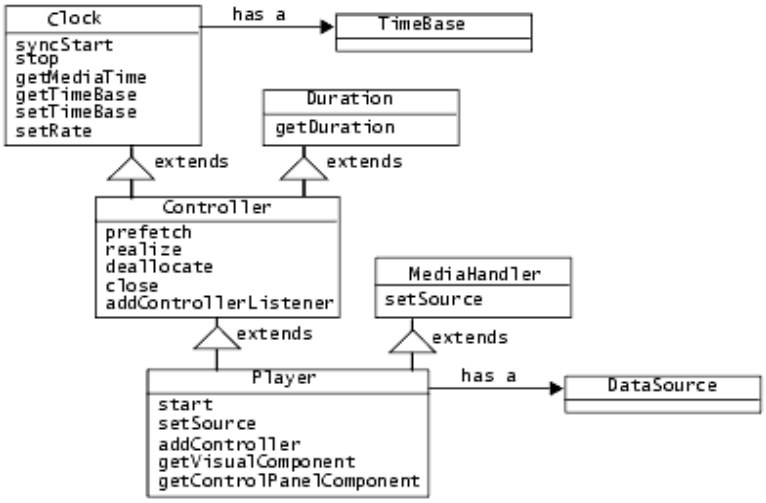
Player

Un Player processa un flusso multimediale in entrata e lo modifica a un determinato tempo. Un DataSource è utilizzato per trasmettere il flusso in entrata al Player e la destinazione dei dati dipende da come questi devono essere rappresentati.

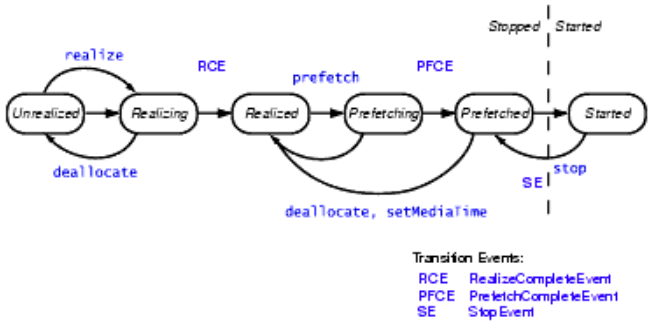


.

I Player non permettono alcun controllo sul processo che stanno eseguendo o su come vengono modificati i dati.



Possono assumere 6 stati diversi. L'interfaccia Clock definisce i primi 2 stati: Stopped e Starter e per facilitare il controllo delle risorse Controller divide lo stato Stopped in 5 stati di standby: Unrealized, Realizing, Realized, Prefetching, and Prefetched.



Analizziamoli uno ad uno:

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() della 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 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 per cui è qui che si rendono disponibili i Control-Panel ed il Visual-Panel. Un player nello stato realized non blocca nessun altro player candidato alla riproduzione del media, per cui possono esistere più player in questo stato.
Prefetching: quando il player è nello stato Realized, è possibile utilizzare il metodo prefetch() per portarsi nello stato di Prefetching. Questo stato è caratterizzato dal fatto che il player inizia ad acquisire le risorse per la riproduzione del media, comprese le risorse esclusive se accessibili. Esso si riserva anche un certo buffer in cui farà il preloading dei primi dati da riprodurre. Il contenuto di questo buffer potrà eventualmente essere visualizzato da un Control-Panel di tipo CachingControl.
Prefetched: terminata la precedente fase, si ha il passaggio automatico allo stato di prefetched. Qui il player è pronto a partire. Tutte le risorse necessarie sono state acquisite ed eventuali altri player che utilizzassero le stesse risorse sono impossibilitati all'esecuzione fino a che le risorse stesse non saranno rilasciate.
Started: finalmente attraverso il metodo start() è possibile iniziare la riproduzione della risorsa attraverso il player.

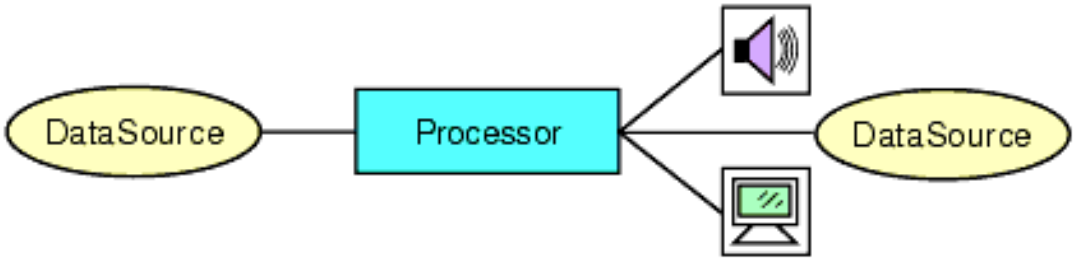
Per il passaggio da uno stato ad un altro esistono cinque metodi principali che possono essere applicati solamente quando il player è in un particolare stato pena la generazione di una eccezione anch'essa esplicativa del problema. Esistono i seguenti cinque metodi:

- start(): appartiene all'interfaccia Clock e può essere eseguito solamente quando il player è nello stato Prefetched. Lo stato seguente è lo stato Started.
- Stop(): appartiene all'interfaccia Clock e può essere eseguito solamente quando il player è nello stato Started. Lo stato seguente è lo stato Prefetched.
- Realize(): serve per portare il player dallo stato di Unrealized allo stato di Realizing.
- Prefetch(): serve per iniziare la fase di prefetching
- deallocate(): serve per liberare le risorse esclusive acquisite dal player. In questo modo un eventuale altro player nello stato di Realized, può eseguire la prefetch().

Il passaggio tra gli stati è notificato attraverso l'emissione di un evento la cui gestione è alla base della programmazione con le JMF.

Processors

Possono essere utilizzati per rappresentare i dati multimediali e sono solamente una specializzazione dei Player che permettono di controllare quali processi vengono eseguiti sui dati. Processor supporta gli stessi controlli di presentazione dei Player e in aggiunta permette di destinare i dati elaborati ad un DataSource cosicché questo può inviarli ad un altro Player, Processor o inviarli a qualche altra destinazione come un File.



Controlli per la presentazione

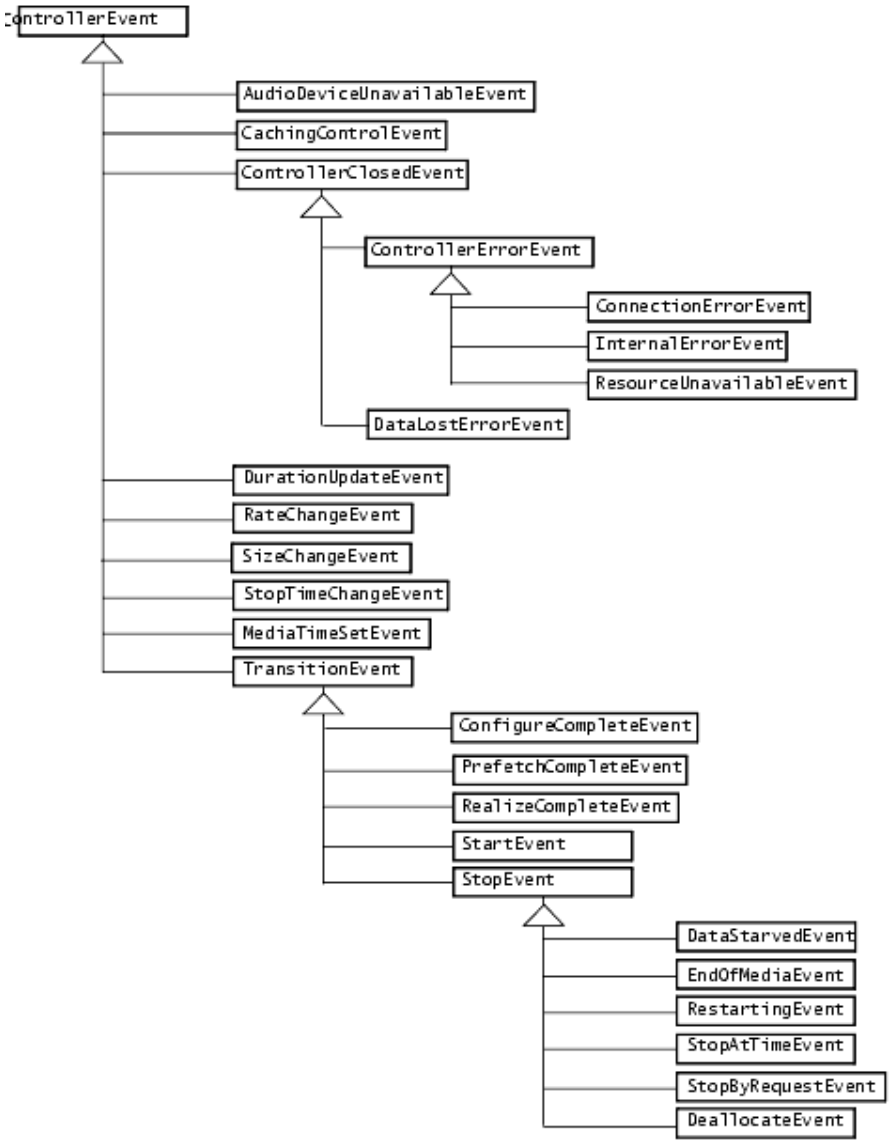
In aggiunta ai controlli standard di presentazine implementati dal Controller, Player e Processor permettono anche di aggiustare il volume della riproduzione. Permettono l'utilizzo di due interfacce di componenti, una componente visuale e una componente control-panel. Possiamo accedere a questi Components direttamente attraverso i metodi getVisualComponent e get-ControlPanelComponent.

Controller Events

ControllerEvents postati da un Controller come Player o Processor cadono in tre categorie: change notifications, closed events e transition events:

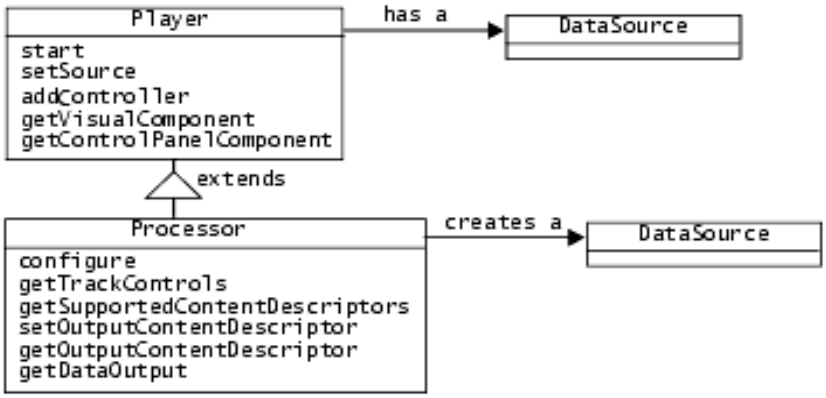
- Change notification: sono i metodi come RateChangeEvent, DurationUpdateEvent e FormatChangeEvent che indicano che qualche attributo del Controller è cambiato; per esempio un Player posta un RateChangeEvent quando il suo rate è cambiato usando il metodo setRate;
- TransitionEvents: permettono al programma di rispondere ai cambiamenti di stato di un oggetto Controller. Un Player posta eventi di transizione ogni volta che si muove da uno stato all'altro;
- ControllerClosedEvents: sono postati da un Controller quando viene chiuso e non è più utilizzabile. I ControllerErrorEvent sono un tipo speciale di questi eventi e vengono

richiamati quando il programma comunica al Controller malfunzionamenti minimizzando l'impatto sull'utente.



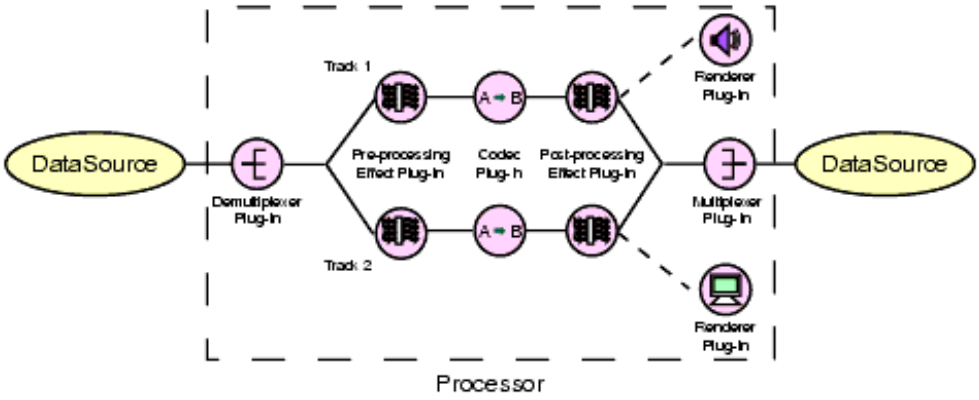
Processi

Un Processor è un Player che prende un DataSource come input, esegue alcuni processi, definiti dall'utente, sui dati multimediali, e come output restituisce i dati modificati.



Può spedire i dati di output ad un dispositivo di presentazione o ad un DataSource che può essere usato come input per Player o Processor o per un altro DataSink.

I processi eseguiti da un Player sono predefiniti e non possono essere monitorati o modificati mentre un Processor permette di definire il tipo di operazioni che possono essere applicate ai dati. Questo permette all'applicazione di applicare effetti e mixare i dati in tempo reale.



Il processo applicato sui dati multimediali è diviso in vari passi:

- Demultiplexing: è il processo di parificazione del flusso in input. Se stiamo trattando un flusso multi traccia ci permette di suddividerlo e mettere in output le tracce separate. Ad esempio un file QuickTime può essere diviso in traccia audio e video. Demultiplexing è eseguito automaticamente ogni volta che il flusso in input contiene dati multipli.
- Pre-Processing: è il processo che applica l'algoritmo degli effetti sulle tracce estratte;
- Transcoding: effettua su ogni traccia l'operazione di codifica o decodifica;
- Post-Processing: applica gli algoritmi degli effetti alle tracce decodificate;
- Multiplexing: unisce le singole tracce modificate in un'unica traccia. Per esempio una traccia video e audio saranno unite in un unico file MPEG. Con il metodo SetOutputContentDescriptor possiamo determinare il tipo del file di output.

- Rendering: è il processo di presentazione del file multimediale all'utente.

Il processo è eseguito ad ogni passo da un diverso componente rappresentato da un JMF plug-in. Ci sono 5 tipi di plug-in:

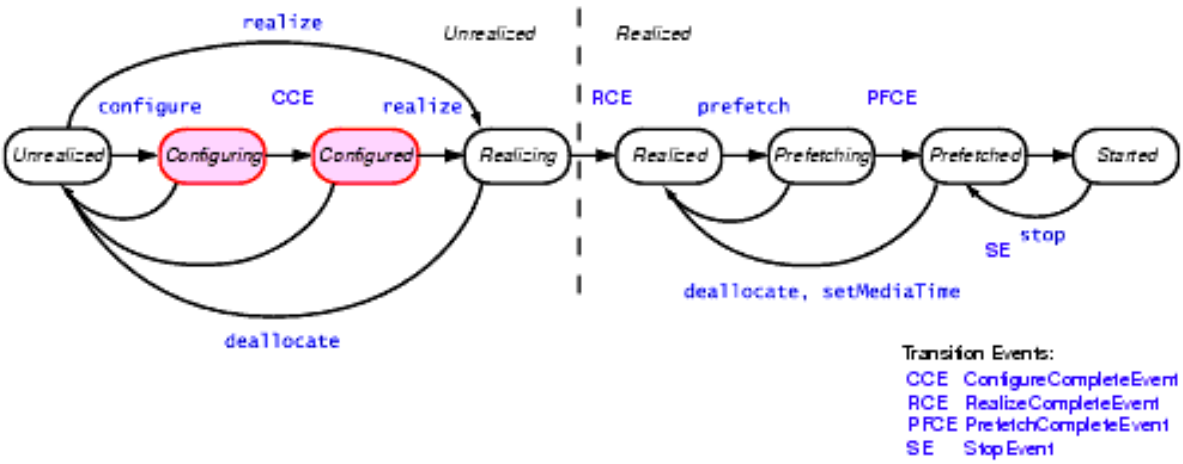
- Demultiplexer: modifica i file di tipo WAV, MPEG o QuickTime, dividendo una traccia una in più tracce.
- Effect: inserisce determinati effetti ad una traccia;
- Codec: effettua codifica e decodifica dei dati;
- Multiplexer: unisce tracce diverse in un'unica traccia;
- Render: elabora il file multimediale e lo invia ad un dispositivo per la riproduzione.

Stati di un Processor

Rispetto ad un Player un Processor ha due stati addizionali che vengono eseguiti prima di entrare nello stato Realizing:

- Configuring: il Processor entra in questo stato quando lo si sta configurando e finché rimane in questo stato si connette al DataSource, separa le tracce multiple e accede alle informazioni riguardanti i dati di input.
- Configured: il Processor entra in questo stato quando è connesso al DataSource e il formato dei dati è stato determinato e posta un ConfigureCompleteEvent.

Quando Realize è chiamato, il Processor transita nelle stato Realized ed è completamente costruito.
Finche il Processor è nello stato Configured, getTrackControls può essere chiamato per avere l'oggetto TrackControl che ci permette di specificare quale operazioni devono essere effettuate sulle varie tracce.
Chiamando Realize direttamente su un Processor non ancora completato, automaticamente transitiamo da Configuring a Configured e poi Realized, non potendo così configurare le operazioni da eseguire con il comando TrackControls e vengono usate le impostazioni di default.



Cattura: Media Data Storage e Transmission

I DataSink sono usati per leggere dati multimediali da un DataSource per poi inviarli a determinate destinazioni. Particolari DataSink permettono di scrivere i dati su file, di inviarli sulla rete o funzionano come trasmettitore RTP. Come i Player gli oggetti DataSink sono costruiti attraverso l'oggetto Manager usando un DataSource e possono usare StreamWriterControl per fornire controlli supplementari sulla scrittura dei dati su file.

Storage Controls

DataSink per informare sul suo stato posta DataSinkEvent di due tipi di versi:

- DataSinkErrorEvent: che indica che è occorso un errore mentre il DataSink stava scrivendo i dati;
- EndOfStreamEvent: che indica che il flusso di dati è stato scritto completamente senza errori.

Per poter rispondere agli eventi postati da un DataSink bisogna implementare l'interfaccia DataSinkListener.

Estensioni

Possiamo estendere le JMF implementando plug-in personalizzati, media handlers, e data sources.

Implementare Plug-Ins

Implementando una delle intefaccie dei plug-in, possiamo accedere e manipolare i dati associati ad un Processor. Le principali interfacce sono:

- Demultiplexer: permette di specificare quante e quali tracce devono essere estratte da un flusso multimediale multitraccia;
- Codec: abilita ad effettuare codifiche e decodifiche dei dati nei formati supportati dalle Jmf API;
- Effect: consente di applicare effetti ai dati multimediali;
- Multiplexer: permette di specificare quante e quali tracce devono essere unite in unico flusso;
- Render: permette di controllare le tracce alle quali è applicato un processo e spedirle ad un dispositivo di output.

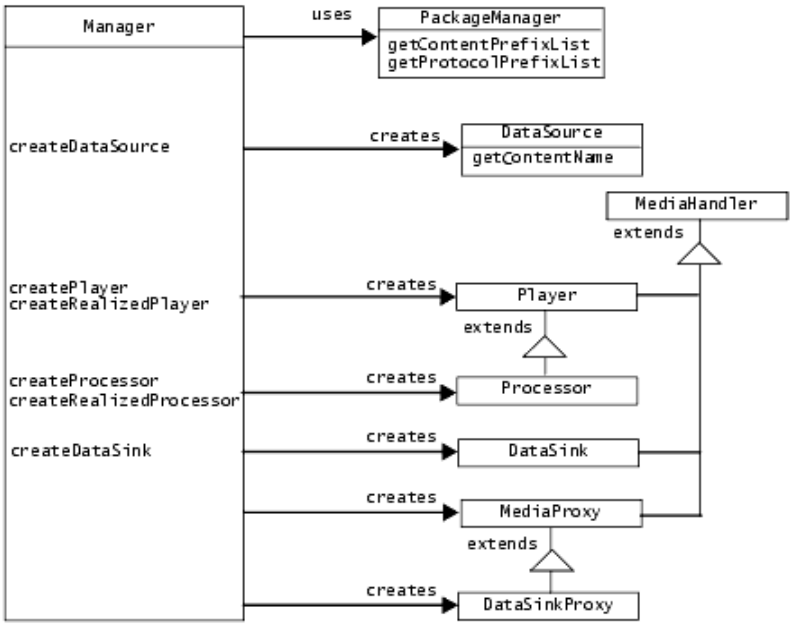
La JMF Plug-In API è una parte delle JMF API, ma Player e Processor non sono richiesti per supportare i plug-in. Nella versione 1.0 delle API questi Plug-in non lavorano con alcuni Player ed alcune implementazione dei Processor possono non supportarli, mentre la versione 2.0 delle API supporta completamente plug-in API.
Codec, Effect e Renders plug-in personalizzati sono accessibili da un Processor attraverso la l'interfaccia TrackControl. Per rendere disponibile ad un Processor o ad un Processor costruito con il ProcessorModel, bisogna registrare il plug-in con il PlugInManager. Una volta registrato può essere utilizzato accedendoci attraverso il Manager quando si costruisce l'oggetto Processor

Implementare MediaHandlers e DataSources

Se i plug-in non sono sufficienti al grado di flessibilità di cui si ha bisogno, si possono implementare direttamente alcune interfacce chiave: Controller, Player, Processor, DataSource, e DataSink.
Il meccanismo usato dal Manager per costruire i vari oggetti permette la costruzione personalizzata di queste interfacce.
La classe personalizzata può essere costruita unicamente se gli è stato assegnato un package univoco con il PackageManager e messa nel posto corretto nella gerarchia dei package.

Costruzione MediaHandler

Players, Processors, e DataSinks sono tutti tipi di MediaHandlers, tutti leggono dati da un DataSource. Un MediaHandler è sempre costruito per una particolare DataSource, che può essere identificato specificatamente o assieme ad un MediaLocator. Quando uno dei metodi createMediaHandler viene chiamato, il Manager usa il nome del content-type ottenuto dal DataSource per trovare e creare l'oggetto MediaHandlers appropriato.



JMF supporta anche un altro tipo di MediaHandler, MediaProxy che elabora contenuti da un DataSource per crearne un altro. Tipicamente, MediaProxy legge un file di configurazione che contiene tutte le informazioni necessarie per fare una connessione ad un server e ottenere i dati. Per creare un Player usando un MediaProxy dobbiamo seguire i seguenti passi partendo da un Manager:

- costruire un DataSource per il protocollo descritto dal MediaLocator;
- usare il content-type del DataSource per costruire il MediaProxy per leggere il file di configurazione;
- Ottenere un nuovo DataSource dal MediaProxy;
- Usare il content-type del nuovo DataSource per costruire il Player;

Il meccanismo che il Manager usa per localizzare e istanziare un appropriato MediaHandler per un DataSource è lo stesso per tutti i tipi di MediaHandlers:

- usando la lista dei package installati dal PackageManager, il Manager genera un lista delle classi MediaHandler disponibili;
- il Manager scorre ogni classe della lista finché non trova una classe nominata Handler che può essere costruita e alla quale può essere collegato il DataSource.

Quando costruiamo Player e Processor, i Manager generano la lista delle handler classi disponibili partendo dalla lista dei content package-prefixes e dai nomi dei content-type del DataSource. Per cercare un Player il Manager cerca le classi con questo prefisso:

<content package-prefix>.media.content.<content-type>.Handler

mentre per un Processor cerca queste classi:

<content package-prefix>.media.processor.<content-type>.Handler

Se il MediaHandler trovato è un MediaProxy, i Managers creano un nuovo DataSource dal MediaProxy e ripetono la ricerca.

Se non vengono trovati dei MediaHandler appropriati, la ricerca viene ripetuta sostituendo il nome del content-type con “unknown”. Il content type unknown è supportato dai Players generici che sono in grado di gestire una grande varitetà di tipi multimediali, spesso in una piattaforma dipendente.

Per far si che un DataSink appena creato invii i dati, letti da un DataSource, la destinazione di output deve essere allegata ad esso. Quando costruiamo un DataSink i Manager usano la lista dei content package-prefixes e il protocollo del Medialocator che identifica la destinazione. Per ogni content package-prefixes il Manager aggiunge alla lista ottenuta dalla ricerca il nome della classe con questa forma:

<content package-prefix>.media.datasink.protocol.Handler

Se il Mediahandler localizzato è un DataSink, il Manager lo istanzia, imposta il suo DataSource e Medialocator e ritorna l’oggetto DataSink ottenuto. Se l’handler è un DataSinkProxy, il Manager riceve il tipo del contenuto del proxy e genera la lista delle classi DataSink che supportano il protocollo del MediaLocator di destinazione e il tipo del contenuto che il proxy restituisce. La forma sarà la seguente:

<content package-prefix>.media.datasink.protocol.<content-type>.Handler

Il processo continua finché un DataSink adatto non viene trovato o il Manager ha controllato tutti i package.

Costruzione DataSource

Il Manager usa lo stesso meccanismo dei MediaHanders per costruire i DataSource, eccetto che genera la lista delle classi dei DataSource partendo dalla lista dei protocol package-prefixes installati.

Per ogni package di protocollo, il Manager aggiunge alla lista di ricerca una classe con questa forma: <protocol package-prefix>.media.protocol.<protocol>.DataSource

Il Manager controlla ogni classe della lista finché non trova un DataSource che può istanziare e al quale può collegare il MediaLocator.

Per rappresentare dati audio e video JMF usa un Player. La riproduzione può essere controllata attraverso il codice oppure mostrando un pannello di controllo. Se si ha il bisogno di riprodurre più flussi multimediali possiamo usare un Player per ogni flusso. Per utilizzarli in modo sincronizzato utilizziamo uno dei Player per controllare le operazioni degli altri.

Un Processor è un Player speciale che permette di controllare quali operazioni devono essere eseguite sui dati prima della loro rappresentazione. I metodi per il controllo della riproduzione dei dati sono gli stessi sia si stia usando un Player, sia si stia usando un Processor.

Il MediaPlayer bean è un Java Bean che incapsula una Player JMF per facilitare la riproduzione di dati multimediali in un applet o applicazione. Il MediaPlayer bean automaticamente costruisce un nuovo Player quando un nuovo flusso multimediale è selezionato e rende facile la riproduzione di una serie di media clips o permette all'utente di decidere quale media clip riprodurre.

2.6.2 Presentazione dai multimediali con Jmf: realizzazione e gestione di player e processor

Controllo dei Player

Per riprodurre un flusso multimediale abbiamo bisogno di costruire un Player, configurarlo, prepararlo per essere eseguito ed infine farlo partire per incominciare la riproduzione.

Creare un Player

Il Player viene creato indirettamente attraverso il Manager multimediale e per visualizzarlo dobbiamo avere un componente costituito dall'oggetto Player che va inserito nell'applet o nell'applicazione a finestra.

Quando si ha bisogno di creare un nuovo Player, viene richiesto al Manager attraverso il metodo createPlayer o createProcessor. Il Manager usa l'URL o il Medialocator specificato per creare il Player appropriato.

Un URL può essere correttamente costruito solo se il corrispondente URLStreamHandle è installato, mentre il MediaLocator non necessita di questa restrizione.

Blocco delle operazioni finche un Player non è realizzato

Molti metodi che possono essere chiamati su di un Player richiedono che quest'ultimo sia stato realizzato e una via per garantire che il Player sia Realized, quando chiamiamo questi metodi, è di usare il metodo createRealizedPlayer. Questo metodo è una strategia conveniente per creare e realizzare un Player in un singolo passo e quando viene chiamato si blocca finchè il Player non entra nello stato Realized. Il Manager fornisce il metodo equivalente createRealizeProcessor per costruire un Processor realizzato.

A volte però questa operazione può creare risultati insaspettati; ad esempio se createRealizedPlayer viene chiamato in un applet, i metodi start e stop dell'applet non sono in grado di interrompere il processo di costruzione.

Utilizzare un ProcessorModel per creare un Processor

Il ProcessorModel definisce i requisiti di input e output per un Processor e il Manager crea un Processor che soddisferà al meglio questi requisiti. Per creare un Processor usando ProcessorModel bisogna chiamare il metodo Manager.createRealizedProcessor. Nell'esempio viene creato un Processor che produce un flusso audio nel formato IMA4 con un campionamento a 16-bit e 44.1 kHz.

```
AudioFormat afs[] = new AudioFormat[1];

afs[0] = new AudioFormat("ima4", 44100, 16, 2);

Manager.createRealizedProcessor(new ProcessorModel(afs, null));
```

Costruire un Processor mediante ProcessorModel.

Finché il ProcessorModel non specifica un URL sorgente, come nell'esempio, il Manager automaticamente trova il primo dispositivo che può catturare un flusso audio e successivamente crea un Processor che lo codifica in IMA4.

Particolare da sottolineare è che la creazione di un Processor con questo metodo non permette di specificare le opzioni dei processi attraverso l'oggetto TrackControls del Processor.

Visualizzare il componente interfaccia multimediale

Un Player ha due tipi di interfaccia utente, una componente visuale e un pannello di controllo.

Visualizzare un Visual Component

È la parte dove avviene la riproduzione del flusso multimediale. Per visualizzare questo tipo di oggetto si deve:

- ottenere il componente chiamando il metodo getVisualComponent;
- aggiungerlo all'applet o all'applicazione;

Possiamo accedere alle proprietà di visualizzazione dell'oggetto Player attraverso la componente visuale e il layout del componente Player è controllato attraverso l'AWT layout manager.

Visualizzare il pannello di controllo

Ogni Player è fornito di un pannello di controllo che permette all'utente di controllare la rappresentazione del flusso multimediale. Per visualizzarlo:

- chiamiamo il metodo getControlPanelComponent per ottenere il Component;
- aggiungiamo il Component ottenuto all'applet o applicazione

Se si preferisce definire un proprio pannello di controllo possiamo implementare una GUI personalizzata e chiamare il metodo del Player appropriato come risposta ad una azione dell'utente. Se il componente personalizzato viene registrato come ControllerListeners, possiamo anche effettuare un aggiornamento su di esso quando lo stato del Player cambia.

Visualizzare il componente Gain-Control.

I Player che permettono degli aggiustamenti sui flussi audio implementano l'interfaccia GainControl che fornisce i metodi per regolare il volume dell'audio come setLevel e setMute.
Per mostrare il GainControl Component, se permesso dal Player bisogna:

1. chiamare il metodo getGainControl dal Player. Se ritorna null questa interfaccia non è supportata;
2. chiamare il metodo getControlComponent sul GainControl ottenuto;
3. aggiungere il Component ottenuto all'applet o applicazione ();

Bisogna osservare che getControls non ritorna un oggetto Player GainControl, ma si può accedere al GainControl solo attraverso la chiamata del metodo getGainControl.

Visualizzare componente di controllo personalizzato

Alcuni tipi Players hanno alcune proprietà che possono essere gestite dall'utente. Per esempio, un video Player può permettere all'utente di aggiustare il contrasto e la luminosità che non sono gestite attraverso l'interfaccia del Player. La lista dei controlli supplementari disponibili è ottenuta attraverso il metodo getControls.

```
Control[] controls = player.getControls();

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

if (controls[i] instanceof CachingControl) {

cachingControl = (CachingControl) controls[i];

}

}
```

Usare il metodo getControls per trovare quali Controls sono supportati.

Settare il rate di riproduzione

L'oggetto Rate di un Player evidenzia quanto il tempo del flusso multimediale cambia rispetto al tempo base definendo di quante unità avanza il flusso per ogni unità di tempo base. L'oggetto rate può essere espresso come fattore a scala temporale, infatti un rate di 2.0 indica che il tempo del flusso multimediale e 2 volte più veloce del tempo reale trascorso dall'avvio del player. In teoria il rate può essere settato ad ogni numero reale, ma alcuni formati multimediali hanno delle dipendenze tra frames che rendono impossibile o impraticabile la riproduzione inversa o ad un rate non standard.

Per settare questo parametro viene chiamato il metodo setRate al quale si passa un valore di tipo float; i Player garantiscono di supportare una rate pari a 1.0.

Impostare la posizione iniziale

Per una sorgente multimediale come un file il tempo è delimitato; il tempo massimo è definito dalla fine del flusso. Per settare il tempo viene usato il metodo setMediaTime e passato un oggetto di tipo Time.

Posizionamento Frame

Alcuni Players permettono di posizionarsi su di un particolare frame di un video, permettendo di impostare la posizione di inizio riproduzione su di un frame senza dover specificare in modo preciso il tempo corrispondente a questa posizione.

Dopo che il frame è stato settato con il metodo FramePositioningControl, il Player imposta il tempo di riproduzione con il valore che corrisponde con l'inizio del frame scelto e un MediaTimeSetEvent viene postato.

Altri Player invece permettono di effettuare conversioni tra tempo di riproduzione e posizione dei frame utilizzando i metodi FramePositioningControl, mapFrameToTime e mapTimeToFrame, ma bisogna ricordare che in certi casi le corrispondenze non coincidono perfettamente.

Preparazione all'avvio

Prima che il Player possa essere avviato, alcune condizione harware e software devono essere soddisfatte. Se il Player non è mai stato utilizzato è necessario che sia allocato un buffer di memoria per memorizzare i dati o se il dato multimediale si trova sulla rete deve essere abilitata una connessione prima di effettuare il download

Realizing e Prefetching un Player

JMF divide il processo di preparazione di un Player in 2 fasi Realizing e Prefetching. Queste due fasi permettono di ridurre i tempi che intercorrono tra l'avvio del Player e l'effettiva riproduzione dei dati. L'implementazione dell'interfaccia ControllerListener permette di controllare quando queste operazione vengono eseguite.

I Processor introducono un'ulteriore fase di Configuring nella quale vengono selezionate le opzioni per controllare in che modo il Processor modificherà i dati multimediali.

Chiamando il metodo realize il Player passa nello stato Realizing ed inizia il processo di realizzazione mentre il metodo prefetch sposta lo stato del Player in Prefetching ed inizia il processo di prefetching. I due metodi sono asincroni e ritornano immediatamente. Quando il Player completa tutte le operazioni posta un RealizeCompleteEvent o PrefetchCompleteEvent.

Un Player nello stato Prefetched è pronto per essere avviato e il suo stato di latenza è notevolemente ridotto, nonostante questo l'esecuzione del metodo setMediaTime può far tornare il Player nello stato Realized incrementando il tempo di latenza.

Bisogna ricordare che lo stato Prefetched tiene occupate le risorse di sistema necessarie ad eseguire le operazioni e nel caso in cui queste possano essere usate da un programma alla volta bisogna evitare che gli altri Players partano.

Avviare la riproduzione

Tipicamente l'avvio di una riproduzione avviene con il metodo Start che informa il Player di avviare la presentazione del dato multimediale il più presto possibile e se è necessario questo metodo prepara il Player a partire eseguendo le operazioni di Realize e Prefetch. Se il metodo è chiamato su un Player in esecuzione l'unico effetto è che un StartEvent è postato in riconoscimento del metodo chiamato.

Clock definisce un metodo syncStart che può essere usato per effettuare delle sincronizzazioni.(vedere Synchronizing Multiple Media Streams).

Per avviare un Player ad un specifico punto in un filmato bisogna:

- specificare il punto nel flusso con il metodo setMediaTime dell'interfaccia Clock;
- chiamare il metodo start sul Player.

Fermare la riproduzione

Ci sono quattro situazioni nelle quali la presentazione verrà fermata:

- quando il metodo stop viene chiamato;
- quando il tempo stop viene raggiunto;
- quando non ci sono più dati da rappresentare;
- quando il dato multimediale è ricevuto troppo lentamente per essere riprodotto.

Quando un Player viene fermato il tempo del flusso multimediale è congelato se la sorgente multimediale può essere controllata, mentre se il Player presenta solamente un flusso non è possibile segnare il tempo. In questo caso solo il ricevente del flusso viene fermato mentre i dati continuano ad essere trasmessi e il tempo avanza.

Al riavvio di un Player se il tempo era stato congelato la presentazione riparte da dove era stata interrotta, mentre nell'altro caso ricomincia con l'ultimo dato ricevuto.

Lo stop di un Player avviene con la chiamata del metodo stop, nel caso in cui il metodo venga chiamato su Player già fermato sarà postato un StopByRequestEvent in rinascimento del metodo chiamato.

Rilasciare le risorse di un Player

Il metodo di deallocazione ordina al Player di rilasciare ogni risorsa esclusiva e minimizzare l'uso delle risorse non esclusive. La gestione della memoria e del bufferring per i Player non sono specificate e normalmente alloca uno spazio di memoria più grande delle dimensioni standard richieste dagli oggetti Java e un Player costruito bene rilascia il maggior quantitativo di memoria

quando il metodo Deallocate viene chiamato. Questo metodo può essere chiamato solo su un Player nello stato Stopped. Per avere un ClockStartedErrors, bisogna chiamare prima il metodo Stop e poi Deallocate, mentre se Deallocate viene chiamato su un Player in Prefetching o Prefetched ritorna allo stato Realized. Se il Player si trova nello stato Realizing e si effettua una chiamata con questo metodo verrà postato un DeallocateEvent evento e ritornerà nello stato Unrealized.
Generalmente questo metodo viene usato quando un Player non viene più usato, anche se il programma mantiene comunque delle referenze ad esso, garantendo così che in un futuro possa essere riavviato velocemente.
Ultima operazione da effettuare è la chiusura del Player con il metodo Close, che informa che il Controller non sarà più usato e potrà essere chiuso rilasciando tutte le risorse che il Controller stava usando e terminando ogni sua attività. Quando un Controller viene chiuso un ControllerClosedEvent è postato e nessun metodo potrà più essere invocato previa la generazione di errori.

Ottenere il Playback Rate

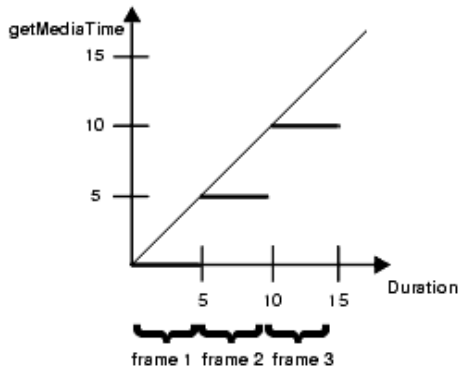
Per ottenere il rate attuale viene invocato il metodo getRate che ritorna un valore tipo float.

Ottenere il Media Time

Si ottiene con il metodo getMediaTime e ritorna un oggetto di tipo Time. Se il Player non sta riproducendo niente, questo è il punto da quale comincerà la presentazione.

Come sappiamo non ci sarà un corrispondenza precisa tra frames e tempo del flusso multimediale, perché i primi rappresentano un intervallo di tempo ben definito mentre il secondo continua ad avanzare durante questo periodo.

Supponiamo di avere uno slide-Show che mostri una slide ogni 5 secondi, che avrà quindi un rate di 0.2 frames al secondo. Se il Player parte al tempo 0.0, sarà visualizzato il primo frame e il tempo avanza da 0.0 a 5.0, mentre se il tempo di partenza è 2.0 il primo frame è visualizzato per 3 secondi, finche 5.0 non è raggiunto.



Ottenere il Tempo Reale di un flusso multimediale

Attraverso l'oggetto TimeBase di un Player e il metodo getTime possiamo ottenere il tempo base di un flusso multimediale:

myCurrentTBTime = player1.getTimeBase().getTime();

Mentre un Player sta funzionando possiamo ottenere il tempo base che corrisponde ad un particolare tempo multimediale chiamando mapToTimeBase.

Ottenere la durata di un Media Stream

Tutti i Controllers implementano l'interfaccia Duration che implementa un unico metodo, getDuration. La durata rappresenta la lunghezza di un flusso multimediale se riprodotto ad un rate 1.0 ed è accessibile solamente attraverso il Player.

Oltre ad un valore determinato questo metodo può ritornare altri 2 valori:

1. DURATION_UNKNOWN: la durata non può essere determinata, ad esempio nel caso in cui il Player raggiunge uno stato dove la durata del flusso non è disponibile; ad un tempo successivo la durata sarà comunque disponibile e potra essere ottenuta con il metodo getDuration;
2. DURATION_UNBOUNDED la sorgente multimediale non ha una durata definita come nel caso in cui si stia acquisendo dati attraverso una webcam o una videocamera.

Implementare l'interfaccia ControllerListener

ControllerListener è un'interfaccia asincrona per catturare gli eventi generati da un oggetto Controller che permette di gestire il tempo che il Player consuma nelle operazioni come il prefetching.

Per implementare questa interfaccia dobbiamo:

- estendere la classe con questa interfaccia;

- registrare la classe come un listener chiamando addControllerListener sul Controller dal quale vogliamo che riceva l'evento.

Quando un Controller posta un evento chiama il controllerUpdate su ogni listener registrato; tipicamente questo metodo implementa una serie di strutture if-else che eliminano gli eventi che non interessano. Nel caso in cui sia stato registrato un listener con più di un Controller, bisogna determinare quale Controller posta l'evento, utilizzando la referenza alla sorgente, fornita dal ControllerEvents, e restituita dal metodo GetSource.
Alla ricezione di un evento da un Controller dobbiamo eseguire dei processi addizionali per garantire che sia nello stato corretto prima di chiamare un metodo di controllo.

```
if (event instanceof EventType){

...

} else if (event instanceof OtherEventType){

...

}
```

Implementazione controllerUpdate.

Alcuni ControllerEvent contengono informazioni addizionali sugli stati, per esempio, le classi StartEvent e StopEvent difiniscono ciascuna un metodo che permette di ricevere il tempo del flusso nel quale l'evento è accaduto.

Utilizzo del ControllerAdapter

Il ControllerAdapter è una classe che implementa il ControllerListener e può essere facilmente estesa per rispondere ad Eventi particolari. Per implementare il ControllerListener con questa nuova classe si deve:

- subclassare ControllerAdapter e sovrascrivere i metodi degli eventi interessati;
- registrare la propria ControllerAdapter classe come un listener per un particolare Controller

Quando un Controller posta un evento, chiama controllerUpdate per ogni listener registrato e il ControllerAdapter automaticamente assegna il metodo al suo appropriato evento, tralasciando gli eventi ai quali non si è interessati.
Il seguente esempio estende ControllerAdapter per creare un Player che automaticamente si sposta all'inizio di un flusso multimediale e dealloca il Player quando raggiunge la fine.

```
player.addControllerListener(new ControllerAdapter() {
public void endOfMedia(EndOfMediaEvent e) {
Controller controller = e.getSource();
controller.stop();
controller.setMediaTime(new Time(0));
controller.deallocate();
}})
```

Uso ControllerAdapter.

Se viene registrato un singolo ControllerAdapter come listerner per più di un Player nell'implementazione dei metodi degli eventi bisognerà determinare quale Player genera l'evento attraverso il metodo getSource, che determina dove ControllerEvent è stato originato.

2.6.2.3 Sincronizzare flussi multimediali multipli

Per sincronizzare la riproduzione di flussi multipli, possiamo allineare i vari Player associando ad essi lo stesso TimeBase. Per fare questo dobbiamo usare i metodi getTimeBase e setTimeBase definiti dall'interfaccia Clock. Per sincronizzare due Player dovremmo allora scrivere
player1.setTimeBase(player2.getTimeBase());
Una volta sincronizzati i Player bisognerà gestire il controllo di ogni Player, ma dato che si tratta di un processo complicato le JMF mettono a disposizione un meccanismo di tipo gerarchico nel quale un Player può assumere i controlli di ogni altro Controller. In questo modo il Player designato controlla in modo automatico gli stati degli altri Controller permettendo di interagire da un singolo punto all'intero gruppo.

Usare un Player per sincronizzare Controllers

Sincronizzare Players direttamente usando SyncStart richiede la gestione di tutti gli stati di tutti i Player sincronizzati. Bisogna controllare ciascuno individualmente, attendere gli eventi e chiamare i metodi di controllo appropriati su di essi. Quando il numero di Player comincia a crescere queste operazioni cominciano a diventare laboriose e JMF ci offre una soluzione che consiste nello scegliere un Player che poi sarà utilizzato per gestire ogni Controller.

Quando si interagirà con un “super” Player, tutte le istruzioni saranno automaticamente passate ai Controllers gestiti. Il meccanismo è implementato esclusivamente attraverso i metodi addController e removeController rendendolo facile e leggero da usare, evitando errori generati dalla gestione individuale di ogni Controller. Quando il primo è chiamato su di un Player lo specifico Controller è aggiunto alla lista dei Controllers gestiti dal Player; al contrario con il metodo removeController il Controller sarà eliminato dalla lista.
Quando un Player assume il controllo di un Controller:

- il Controller sostituisce il suo tempo base con quello del Player;
- alla durata e alla latenza dell’oggetto Player saranno aggiunte quelle del nuovo Controller.

Aggiungere un Controller

Per aggiungere un Controller ad un Player usiamo, come già sappiamo, il metodo addController e una volta che è stato aggiunto non potremo più chiamare su di esso alcun metodo, ma potremo farlo solo attraverso il Player che lo gestisce. Per essere aggiunto questo deve essere nello stato Realized, altrimenti si verifica un errore del tipo NotRealizedError. Due Player non possono essere posti uno sotto il controllo dell’altro allo stesso tempo.

Controllare insiemi di Controller

Per controllare le operazioni di un gruppo di Controller interagiamo direttamente con il Player; prima di chiamare un metodo si assicura che il Controller sia nello stato corretto.

Rimuovere un Controller

Usiamo il metodo removeController per rimuovere un Controller dalla lista di un Player.

Sincronizzare i Player direttamente

In alcune situazioni si è interessati a gestire dei Player sincronizzati individualmente e per fare questo bisogna:

- registrare come listener ogni Player sincronizzato;
- scegliere qual è il Player principale e settare il tempo base. Nel caso in cui un Player utilizzi un push-datasource questo dovrà gestire tutti gli altri;
- settare il rate per tutti i Player. Nel caso in cui non sia accettato il Player ritorna il proprio;

- sincronizzare gli stati di tutti i componenti;
- sincronizzare le operazioni per gli oggetti Player:
- settare il tempo multimediale di ognuno;
- Prefetch ogni Player;
- determinare la latenza massima di avvio.

A questo punto bisogna ascoltare gli eventi di transizione per tutti gli oggetti Player e prendere traccia di quelli che hanno postato un evento.
In alcune situazioni bisogna essere certi della risposta agli eventi da parte dei Player sincronizzati ed è necessario che tutti giungano allo stato desiderato.
Per esempio, assumiamo che si stia usando un Player che gestisce un gruppo di altri Player. In un primo momento settiamo il tempo multimediale a 10, avviamo il Player e successivamente cambiamo il tempo a 20. I passi da effettuare sono i seguenti:

- sincronizzare tutti i Player con il metodo setMediaTime con un valore pari a 10;
- chiamare prefetch su ogni Player per prepararli ad essere avviati;
- avviare i Player;
- stoppare tutti i Player quando la richiesta della seconda modifica viene ricevuta;
- effettuare la seconda modifica con il metodo setMediaTime;
- riavviare il prefetching;
- quando tutti i Player hanno raggiunto lo stato prefetched, avviarli con syncStart.

In questo caso aspettare l'evento PrefetchComplete per tutti i Player prima di chiamare syncStart non è sufficiente, perchè non possiamo sapere se questo evento è stato postato per la prima o la seconda operazioni di prefetch. Possiamo risolvere il problema bloccando il processo finché tutti i Player non abbiamo postato l'evento collegato allo stop, garantendo in questo modo che l'evento PrefetchComplete ricevuto è collegato alla seconda operazione di prefetch.

2.6.3 Elaborare dati multimediali con le API JMF

Un Processor, come sappiamo, può essere usato come un Player programmato che permette di effettuare particolari operazioni di codifica e decodifica, multiplexing e rendering dei dati. Possiamo controllare ciò che un processo esegue in diverse maniere:

- usando un ProcessorModel per costruire un Processor che ha caratteristici tipi di input ed output;

- usando il metodo setFormat per specificare quale formato di conversione deve essere applicato su ogni singola traccia;
- usando il metodo setOutputContentDescriptor per specificare quale sia il formato del flusso di output ottenuto dall'unione di più tracce;
- usando setCodecChain per selezionare gli Effect o Codec plug-ins che sono usati dal Processor;
- usando il metodo setRenderer per selezionare il plug-in Render che il Processor deve utilizzare.

Con il metodo di configurazione un Processor passa dallo stato Unrealized allo stato Configuring ed acquisisce tutte le informazione necessarie per la creazione degli oggetti TrackControl di ogni traccia.
Quando ha terminato questa fase passa in Configured, posta un ConfigureCompleteEvent ed è possibile settare il formato di output e le opzioni dei trackControl. Finita la specifica delle carattestiche dei processi, possiamo realizzarlo muovendolo nello stato Realizing. A processo realizzato una modifica alle opzioni non garantisce che lavori correttamente ed un'eccezione del tipo FormatChangeException è sollevata.

Selezionare le opzioni delle tracce

Per selezionare quali plug-ins devono essere usati su ogni traccia di un flusso bisogna:

- chiamare il metodo PlugInManager.getPlugInList per ottenere la lista dei plug-ins compatibili con i formati di input ed output e il loro tipo;
- chiamare il metodo getTrackControls sul Processor per ottenere il controllo su ogni traccia del flusso. In questa fase il Processor deve essere nello stato Configured;
- chiamare i metodi setCodecChain o setRenderer per specificare il plug-in che si vuole utilizzare.

Quando viene usato il metodo setCodecChain per specificare il plug-in del codec e degli effetti, l'ordine nel quale i plug-ins appaiono nel processo è determinato dal formato di input ed output che ogni plug-in supporta.
Per controllare la trascodifica che viene effettuata da un particolare Codec, possiamo usare il metodo getControls che ritorna tutti i controll associati alla traccia, includendo i controlli dei codec come H263Control, QualityControl e MPEGAudioControl.

Convertire dati multimediali da un formato ad un altro

Per selezionare il formato di una particolare traccia attraverso TrackControl bisogna utilizzare getTrackControls sul Processor per ottenere il controllo su ogni traccia del flusso e successivamente chiamare il metodo setFormat per specificare il formato verso il quale si vuole convertire il flusso.

Specificare il formato di output

Attraverso il Processor può essere definito, con il metodo setContentDescriptor, il formato del dato di output, ottenendo preventivamente la lista di tutti i formati supportati con il metodo getSupportedContentDescriptors. Il formato di uscita può anche essere selezionato attraverso il ProcessorModel che crea il Processor.

Specificando automaticamente il formato di output vengono selezionate le opzioni di default del processo per questo formato e sovrascritte tutte le opzioni selezionate precedentemente dal TrackControls.

Impostando il formato di output a “null”, i dati multimediali saranno convertiti nonostante l'output al Processor è l'output dell'oggetto DataSource.

Specificare la destinazione di un dato

È possibile specificare una destinazione per un flusso multimediale selezionando un particolare Render per una traccia attraverso il TrackControl, usando il formato di Output di un Processor come input per un DataSink o usando l’output di un Processor come input per un altro Controller che ha un destinazione differente.

Selezionare un Render

Per selezionare un Render dobbiamo:

- ottenere il controllo sulla traccia;
- chiamare il metodo setRender per specificare il Render plug-in da utilizzare.

Scrivere dati multimediali su file

Può essere utilizzato un DataSink per leggere i dati da un oggetto Processor attraverso un DataSource e scriverli su file.

I passi da seguire sono i seguenti:

- ottenere, mediante un Datasource, con il metodo getDataOutput l’output di Processor;

- costruire un Datasink che ci permetta di scrivere i dati su di un file con Manager.createDataSink passandogli il DataSource e il MediaLocator, che specifica la posizione del file su cui andranno scritti i dati;
- aprire il file chiamando il metodo open sul DataSink;
- avviare il DataSink per far partire la scrittura dei dati.

Il formato dei dati scritti viene controllato attraverso il Processor e di default esso corrisponde al tipo RAW. Per cambiare il tipo del dato di uscita di un DataSource viene usato il metodo setContentDescriptor.

```
DataSink sink;

MediaLocator dest = new MediaLocator(file://newfile.wav);

try{

sink = Manager.createDataSink(p.getDataOutput(), dest);

sink.open();

sink.start();

} catch (Exception) {}
```

Uso di un DataSink per la scrittura su file

Un Processor può abilitare il controllo, da parte dell'utente, di quanti byte possono essere scritti sulla sua destinazione implementando StreamWriterControl. Per verficare se un Processor supporta questo funzione chiamiamo getControl("javax.media.datasink.StreamWriterControl") su di esso.

Connettere un Processor ad un altro Player

L'output di un Processor può essere inviato come input ad un altro Player facendo ritornare un DataSource con il metodo getDataOutput e costruendo con questo un altro Player o Processor attraverso il Manager.

2.6.4 Catturare dati multimediali con le API JMF

JMF può essere usata per acquisire dati multimediali da un dispositivo di acquisizione come un microfono o una videocamera.

Per catturare i dati bisogna:

- localizzare il dispositivo che vogliamo usare facendo un'interrogazione al CaptureDeviceManager;
- ottenere l'oggetto CaptureDeviceInfo per il dispositivo;
- ottenere un MediaLocator dall'oggetto CaptureDeviceInfo e usarlo per creare un DataSource;
- creare un Controller usando il DataSource;
- avviare il Controller per iniziare il processo di acquisizione.

Quando usiamo un DataSource di cattura con un Player possiamo solo riprodurre i dati acquisiti, mentre per poter effettuare delle operazioni sui dati dobbiamo utilizzare un Processor.

Accedere ai dispositivi di cattura

Attraverso il CaptureDeviceManager accediamo ai dispositivi di acquisizione e con il metodo *CaptureDeviceManager.getDeviceList* otteniamo la lista di tutti i dispositivi disponibili alla JMF reperibile con il comando.

Ogni dispositivo è rappresentato da un oggetto di tipo CaptureDeviceInfo che otteniamo con il metodo CaptureDeviceManager.getDevice

CaptureDeviceInfo deviceInfo = CaptureDeviceManager.getDevice("deviceName");

Catturare i dati

Per catturare i dati da un particolare dispositivo, bisogna ottenere il MediaLocator dal suo oggetto CaptureDeviceInfo. Possiamo anche utilizzare questo MediaLocator per costruire un Controller o usarlo per costruire un DataSource che useremo come input per un altro Player o Processor.

Permettere all'utente di controllare il processo di acquisizione

Un dispositivo di cattura generalmente ha una serie di attributi specifici che possono essere modificati in modo da permettere la sua gestione. I controlli possibili sono PortControl e

MonitorControl accessibili attraverso il metodo getControl sul DataSource passandogli il nome del controllo che vogliamo.
Il primo permette di selezionare da quale porta i dati devono essere catturati, mentre MonitorControl fornisce i metodi per monitorare ciò che viene acquisito da un dispositivo ed è particolarmente utile nel caso in cui i dispositivi hanno un monitor per vedere/ascoltare il processo di cattura o codifica.
Se i dispositivi dispongono di altri oggetti di controllo simili a quelli appena descritti questi possono essere ottenuti attraverso il metodo getControlComponent.
Aggiungendo il Component all'applicazione o all'applet possiamo permette all'utente di interagire con il dispositivo attraverso i controlli di acquisizione e possiamo inoltre visualizzare il pannello di controllo standard e componenti visuali associati al Controller che stiamo usando.

```
Component controlPanel, visualComponent;
if ((controlPanel = p.getControlPanelComponent()) != null)
add(controlPanel);
if ((visualComponent = p.getVisualComponent()) != null)
add(visualComponent);
```

Visualizzare GUI per un Processor

Memorizzare i dati acquisiti

Il processo di memorizzazione dei dati su di un file è uguale a quello già illustrato in precedenza con l'aggiunta di alcuni accorgimenti che permettono di acquisire e salvare i dati contemporaneamente e terminare il processo nel momento in cui non verranno più acquisiti dati dal dispositivo.
I passi sono i seguenti:

- ottenere, mediante un Datasource, con il metodo getDataOutput l'output di Processor;
- costruire un Datasink che ci permetta di scrivere i dati su di un file con il metodo Manager.createDataSink passandogli il DataSource e il MediaLocator, che specifica la posizione del file su cui andranno scritti i dati;
- aprire il file chiamando il metodo open sul DataSink;
- avviare il DataSink per far partire la scrittura dei dati;
- avviare il Processor per iniziare la cattura;

- aspettare un evento del tipo EndOfMediaEvent, che può accadere ad un determinato istante o per un'azione da parte dell'utente;
- fermare il Processor alla fine della cattura;
- chiudere il Processor;
- quando il Processor è chiuso e il DataSink posta un EndOfStreamEvent chiudere anche quest'ultimo.

```
DataSink sink;
MediaLocator dest = new MediaLocator("file://newfile.wav");
try {
sink = Manager.createDataSink(p.getDataOutput(), dest);
sink.open();
sink.start();
} catch (Exception) {}
```

Salvare i dati acquisiti su di un file

3. CODEC MULTIMEDIALI & INTERFACCE

In questo capitolo vengono analizzati i codec e le interfacce scelte per l'implementazione delle applicazioni.

3.1 Digital Video

Il formato DV è stato creato per semplificare ed unificare il processo di ripresa, acquisizione, editing, trasferimento e per aumentare la qualità del video rispetto ai sistemi precedenti. Il DV è un formato digitale nativo cioè all'atto della registrazione il video viene registrato in formato binario.

Infatti quando lavoriamo da analogico, ovvero catturiamo una sequenza con una scheda di acquisizione (M-Jpeg, mpeg1/2, ecc...), non facciamo altro che trasformare un segnale analogico che esce dalla videocamera (VHS) in un segnale digitale. Con il DigitalVideo[12] questo non avviene perché già sulla cassetta il filmato è in formato digitale e dobbiamo semplicemente trasferirlo su computer senza operazioni aggiuntive. Un file DV potrebbe presentarsi nel formato AVI o QuickTime, ma ci si deve ricordare che rimane un DV, con una sua struttura e una propria compressione d'immagine.

Risoluzione	720X576 (PAL) - 720X480 (NTSC)
Date Rate	3.6 Mbyte/sec
Frame Rate	25 fps (PAL) - 29.97 (NTCS)
Audio	48 Khz 16 bit /44.1 Khz 16 bit / 32 Khz 12 bit
Fattore di compressione	5:1

Osservando la tabella notiamo che la codifica DV necessita di un grande spazio di memoria per memorizzare i filmati registrati, circa 12 Gbyte per ora; questo perché l'algoritmo di compressione utilizzato è il DCT (Trasformazione Discreta di Coseno) dove ogni frame viene compresso come immagine a sè stante senza essere collegato ai frame successivi o precedenti, rendendo possibile la compressione 5:1.

Per quanto riguarda l'audio vi è la possibilità di scegliere tra 3 diversi tipi di campionamento. I più usati sono i 48 Khz a 16 bit su 2 canali digitali stereo e i 32 Khz a 12 bit su 4 canali.

3.2 FireWire

La grande quantità di dati da trasferire utilizzando il formato Dv ha portato alla necessità di utilizzare un protocollo di trasmissione particolare: il FireWire che non fa parte delle specifiche DV, ma si è sviluppato di pari passo con esso.

Il FireWire è di proprietà della Apple[13] ed è stato sviluppato in principio per essere utilizzato nei personal computer e nei dispositivi multimediali. Attualmente però viene utilizzato per collegare dispositivi di archiviazione di massa o dispositivi di acquisizione video.

In pratica le schede FireWire sono delle interfacce che permettono il trasferimento di dati ad alta velocità tra le periferiche connesse, così come accade con le porte seriali o USB, ma differiscono notevolmente dalle schede di acquisizione analogiche che non servono semplicemente come via di comunicazione, ma devono anche occuparsi della digitalizzazione del segnale video da esse proveniente.

Le velocità di trasmissione che si possono raggiungere utilizzando questo protocollo sono dell'ordine 400 Mbit/sec.

3.3 Ac-3

Rappresenta l'algoritmo di compressione audio nel Dolby Digital[17] dove la codifica avviene con perdita di informazioni come nella codifica MP3 e WMA.

Il sistema Dolby Digital nella versione più diffusa utilizza 5.1 canali audio (5 effettivi e 1 per le basse frequenze) e partendo dal presupposto che un segnale audio stereo lineare a 48 kHZ/16 bit richieda 768 kbps per la sua codifica, otteniamo che per la codifica a 5.1 canali avremmo bisogno di 3840 kbps. Questo porta ad una massiccia richiesta di memoria, ma grazie alla codifica AC-3 siamo in grado di codificare il segnale audio a circa 400 kbps.

L'algoritmo di compressione prende in input l'audio lineare, lo analizza e scarta parte dei dati riducendo la risoluzione effettiva e quindi la dimensione finale, portando però un innalzamento del livello di rumore digitale.

Tanto maggiore è il fattore di compressione, tanto maggiore è la probabilità che il livello di rumore introdotto a causa della perdita di informazioni diventi udibile. Per questo motivo, tanto maggiore è il bitrate utilizzato per la codifica tanto migliore sarà l'aderenza alla qualità dell'audio originale non compresso.

3.4 XviD Mpeg 4

XviD[14] è un codec openSource aderente allo standar MPEG-4[1] originariamente basato su OpenDivx.

Il codec XviD ha avuto una considerevole diffusione soprattutto nell'ambito del filesharing, dove viene utilizzato per comprimere la parte video dei film, cercando di mantenere una buona qualità, in modo che possano occupare poco spazio ed essere trasferiti velocemente nelle reti di file sharing.

3.5 Avi

Acronimo di Audio Video Interleave, è un formato multimediale contenitore realizzato dalla Microsoft nel 1992 come formato standard video per Windows[15]. Questo significa che il flusso video e il flusso audio sono indipendenti e, durante la riproduzione del filmato, vengono eseguiti contemporaneamente in modalità sincrona (in caso di disincronia il video viene rallentato o velocizzato per essere sempre sincronizzato con l'audio).

[1] []Mpeg-4 è uno ISO/IEC sviluppato dalla Moving Picture Export Group che include tutte le caratteristiche degli standard MPEG-1 e MPEG-2 più tutta una nuova serie di caratteristiche come la gestione tridimensionale degli oggetti. Grazie a questo nuovo codec i flusso audio e video possono essere manipolati e modificati in tempo reale.

4. IMPLEMENTAZIONE APPLICAZIONI

In questo capitolo è illustrato lo sviluppo dei processi soggetti a modifica.

4.1 Requisiti non funzionali

In questa sezione vengono riportati i requisiti non funzionali che coprono un aspetto molto importante nello sviluppo dei processi, in quanto il progetto L.O.D.E. è nato in primo luogo con lo scopo di migliorare in maniera drastica i requisiti non funzionali di ePresence che portarono a grosse difficoltà nel suo utilizzo.

4.1.1 Semplicità d'uso

Le applicazioni devono essere semplici da utilizzare in modo da non richiedere particolari conoscenze tecniche.

4.1.2 Robustezza

Le interfacce devono essere semplici ed intuitive in modo da guidare l'utente attraverso i passi dei processi, devono impedire agli utenti di commettere errori.

4.1.3 Automazione

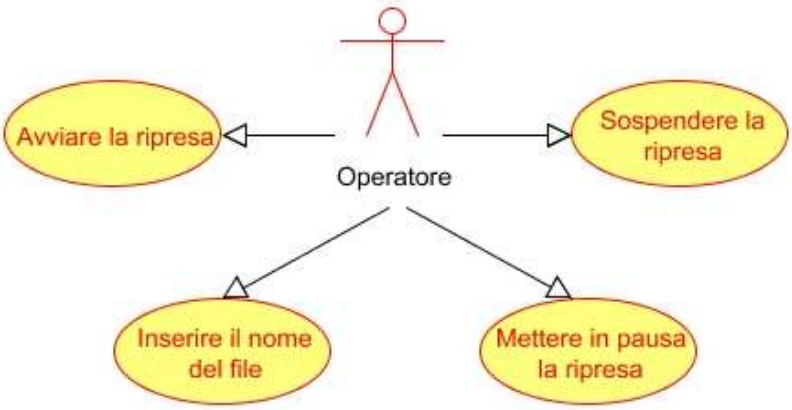
Le applicazione devono richiedere l'intervento dell'operatore solo quando strettamente necessario; il livello di automazione deve essere massimo.

4.2 Requisiti funzionali

In questa sezione vengono elencati i requisiti funzionali del sistema mediante diagrammi dei casi d'uso.

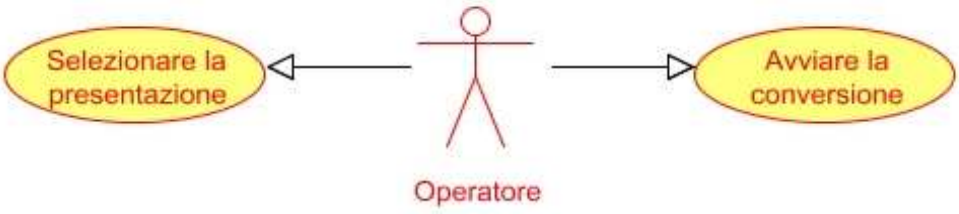
4.2.1 Acquisizione video

È composta dall'insieme delle attività che l'operatore compie nella fase di registrazione video. In questa fase l'operatore può scegliere il nome del file video della lezione e può avviare, mettere in pausa e terminare la registrazione.



4.2.2 Conversione delle presentazioni

In questa fase le presentazioni fornite dai docenti vengono convertite in serie di immagini. L'operatore seleziona la presentazione e avvia il processo di conversione.



4.2.3 Conversione del filmato

Il processo di conversione dei filmati, nella futura versione di L.O.D.E., verrà incluso nella fase chiamata PostProcessing nella quale vengono creati i file utilizzati per la distribuzione delle lezioni via web e su cd. Questa operazione non è un processo a sé stante, ma fa parte di un insieme di operazioni sulle quali l'operatore non interviene direttamente e di conseguenza non esiste un caso d'uso specifico che la rappresenti.

4.3 Processo di acquisizione dei filmati

Il processo di acquisizione è composto da una prima fase di preview, per poter impostare correttamente l'inquadratura della videocamera e da una seconda fase nella quale viene salvato su file quanto viene ripreso. È inoltre possibile mettere in pausa, riprendere e sospendere la registrazione.

L'hardware necessario per l'acquisizione è composto da un pc e da una videocamera digitale collegata alla macchina mediante presa FireWire. All'interno del progetto L.O.D.E. la preview avviene utilizzando un grafo costruito con i filtri DirectShow che cattura il flusso proveniente dalla videocamera e mediante un filtro permette la visualizzazione del suo contenuto creando la preview. La fase di registrazione video viene eseguita anch'essa utilizzando un grafo avente una costruzione un po' più complessa del precedente, in quanto devono essere aggiunti i filtri per la codifica audio e video e per il salvataggio su file del filmato.

Per quanto riguarda l'implementazione in Delphi non si sono dovuti escogitare particolari stratagemmi in quanto l'unica cosa necessaria era, in tutti e due i casi, quella di caricare e far partire il grafo, lasciando eseguire a quest'ultimo tutte le operazioni per il riconoscimento della videocamera e cattura dei flussi video e audio. Per sospendere e stoppare la ripresa è bastato semplicemente, sempre via codice Delphi, mettere in pausa o stoppare il grafo.

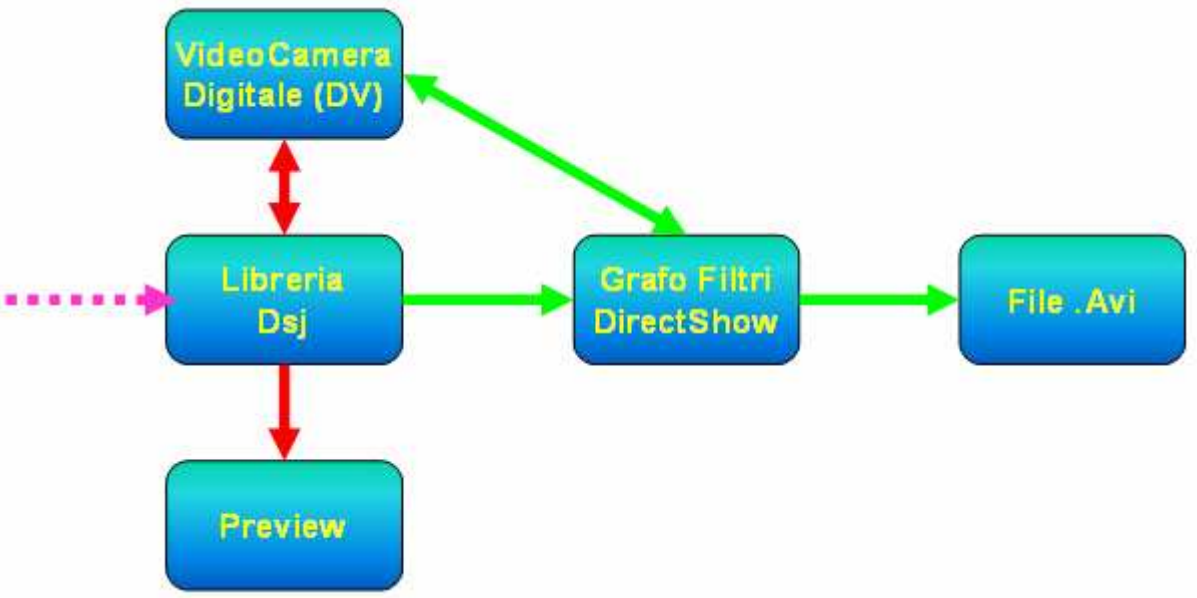
Per l'implementazione in Java come primo approccio si è tentato di utilizzare esclusivamente le API JMF, ma se con dispositivi connessi al pc mediante presa USB non vi era alcun intralcio, utilizzando la presa DV sorgevano problemi in quanto per poter utilizzare questo protocollo in Java è necessario installare sulla macchina la libreria jLibDC1394[11], che è utilizzabile per ora solo sotto Linux.

Escludendo queste API si è reso necessario l'utilizzo di un altro strumento che potesse svolgere questo compito e dopo una lunga ricerca sul web il progetto Dsj fu la soluzione adeguata.

Analizzando il prodotto è risultato possibile abbinarlo con le API JMF, infatti Dsj offre la possibilità di creare un DataSource, rappresentante la videocamera, che viene poi utilizzato per la realizzazione di un Player o Processor, che si occupano della riproduzione e salvataggio del filmato. La soluzione sembrava funzionasse, ma si è visto che i tempi per il riconoscimento del dispositivo di cattura e proiezioni delle immagini erano superiori rispetto ad una soluzione che utilizzasse esclusivamente il progetto Dsj.

Abbandonata definitivamente la possibilità di utilizzare le API JMF si è passati alla definizione della struttura del processo di acquisizione stabilendo che la fase di preview avvenisse prelevando direttamente il flusso multimediale proveniente dalla presa FireWire, mentre per la fase di salvataggio del video si è deciso di utilizzare un grafo creato con i filtri DirectShow.

Il passo successivo è stato quello di creare con GraphEdit il grafo, cercando la combinazione di filtri che permettesse di ottenere un filmato intermedio, che sarebbe poi passato alla fase di conversione, avente le stesse caratteristiche di quelli ottenuti con L.O.D.E. Avendo Dsj la possibilità di eseguire grafi, e quindi di utilizzare i filtri DirectShow, offre un grande vantaggio nel senso che dal lato Java ci si deve solo preoccupare di eseguire il grafo, mentre la ricerca del dispositivo di acquisizione, la codifica dell'audio e video nel formato desiderato e il salvataggio su file del filmato vengono eseguiti dalle Microsoft's DirectShow API .



Struttura processo acquisizione video

Attualmente il principale limite di questa struttura sta nel fatto che se si dovesse usare una videocamera diversa da quella che è stata inserita durante la costruzione del grafo, bisognerebbe ridefinirlo completamente con la nuova videocamera, perchè il grafo ha un struttura statica che non è in grado di riconoscere dispositivi diversi da quello originario, anche se della stessa marca e modello.

Per ovviare a questo problema si sono presentate 3 possibili soluzioni da testare. La prima consiste nell'utilizzare un grafo salvato in un file .xgr (Xml) invece che .grf, in quanto con questo tipo di estensione si è in grado di modificare quei valori che identificano in modo univoco i dispositivi di acquisizione video e audio. La seconda strada è quella di creare il grafo direttamente da codice Java, verificando di volta in volta i dispositivi connessi al pc ed infine l'ultima soluzione, sempre utilizzando codice Java,è quella di connettere "al volo" tutte le componenti necessarie.

L’utilizzo di file .xgr non presenta nessuna complicazione a livello d’implementazione, ma all’esecuzione dell’applicazione la preview del video ripreso non è fluida rendendo impossibile capire cosa si sta riprendendo.
La costruzione del grafo via codice inizialmente sembrava la strada più facile in quanto bisognava solamente prendere i filtri utilizzati per la costruzione del grafo in GraphEdit e connetterli tra loro. Lo sviluppo però si è arenato quando ci si è accorti che Dsj non permette di accedere a tutte le informazioni dei filtri, in quanto molti di essi quando vengono connessi tra loro modificano le loro proprietà, oppure necessitano di un filtro intermedio che viene aggiunto automaticamente.
Abbandonate queste due soluzioni tutte le speranze sono state riposte nell’ultima possibilità che si è dimostrata la soluzione vincente.
Nell’implementare questa nuova versione si è cercato di conservare il più possibile la struttura della versione precedente in modo da rendere facile e veloce l’aggiornamento del codice di L.O.D.E.
All’avvio dell’applicazione vengono ricercati tutti i dispositivi di cattura video e audio connessi al pc ed è data all’utente la possibilità di scegliere con quali di essi eseguire la registrazione ed una volta selezionati viene lanciata la preview per impostare una corretta inquadratura. Le possibili soluzioni, come già visto in precedenza, sono acquisire audio e video dalla videocamera digitale oppure ottenere il video sempre da quest’ultima e l’audio mediante l’utilizzo di una scheda audio.
Il passo successivo, senza alcun intervento dell’utente, è quello di cercare tutti i codec audio e video installati sulla macchina e selezionare i più adatti. Vengono sempre utilizzati l’Xvid per il video e l’AC-3 per l’audio con un campionamento ad 1 canale, sampleRate 8000, 16 bit e 128 kbps.
In un futuro potrà essere l’utente a scegliere il formato audio e video più adatto alle proprie esigenze, ma al momento si è scelto di non fornire questa possibilità in quanto non si è a conoscenza di come l’applicazione risponda con i tutti vari codec (tempi di codifica, allineamento audio e video) e perché, essendo il progetto realizzato esclusivamente con strumenti open-source, anche i codec devono avere la stessa origine.
A questo punto una volta identificato il file di destinazione è possibile avviare la registrazione e avendo mantenuto la struttura precedente è rimasta la possibilità di mettere in pausa e arrestare la registrazione salvando il video su file.
Al momento questa versione è in fase di testing per verificare che funzioni correttamente con tutto il pacchetto L.O.D.E. ed in tempi successivi verrà distribuita.

4.3.1 Realizzazione in Java

L'applicazione è fornita di una semplice interfaccia grafica composta da un frame contenente un Jpanel per la riproduzione di quanto viene acquisito dalla videocamera e da un altro Jpanel contenente i bottoni per avviare, sospendere, terminare l'acquisizione.

Interfaccia processo acquisizione

Il metodo attorno al quale gira tutta l'applicazione è loadMovie(String path, int flags) che riceve in ingresso due parametri, uno per generare la sottoclasse necessaria per svolgere le operazioni desiderate e l'altro per impostare il rendering. Dopo aver eliminato eventuali oggetti DSFiltergraph creati in precedenza viene richiamato il metodo “DSFiltergraph.createDSFiltergraph(path, flags, this)” che genera la sottoclasse corrispondente alla variabile path passata come parametro e viene creato un oggetto DSFiltergraph.DSAudioStream inizializzandolo richiamando sull'oggetto DSFiltergraph il metodo getAudioStream(). In queto modo si ottiene il flusso audio trasmesso da Dsj quando nel costruttore delle sottoclassi il bit riguardante il rendering è settato su ” DELIVER_AUDIO”. Se l'oggetto DSAudioStream non è nullo viene istanziato l'oggetto JaudioPlayer() ed inizializzato con il DSAudioStream. Infine si richiama il metodo initDisplay() e si fa partire il JaudioPlayer() se non è nullo. Il metodo initDisplay() aggiunge solamente al pannello del video il video.

All'avvio dell'applicazione un JOptionPane richiede l'inserimento del nome con cui si vuole salvare il file e successivamente appare la finestra per la gestione delle operazioni di acquisizione con la

preview della videocamera. Come abbiamo detto il flusso della preview proviene direttamente dalla videocamera, utilizzando il metodo “void loadMovie(String path, int flags)” passandogli come parametro la stringa “dv”, per indicare che si vuole creare la sottoclasse DSDVCam che ci permetterà di usare la videocamera digitale come sorgente. Il secondo parametro è “DSFiltergraph.HEADLESS” che rappresenta la modalità di rendering dove Dsj non interferisce in nessun modo con il flusso multimediale eccetto per i controlli base (start, stop, etc.).

Successivamente con la pressione del tasto “Start” se la variabile intera “check” ha valore 0, viene settata ad 1 e lanciato il metodo “loadmovie(path,flags)” passandogli come primo parametro il percorso assoluto del grafo che esegue la registrazione del filmato e come secondo il render “DSFiltergraph.DELIVER_AUDIO”. Nel caso in cui il valore della variabile “check” fosse 1 viene solamente riavviata la registrazione precedentemente messa in pausa con il bottone “Pause”.

Con la pressione del bottone “Stop” si arresta la registrazione, viene rimosso, eliminato l'oggetto DSFiltergraph, il file ottenuto viene rinominato con il nome scelto precedentemente e si esce dal programma.

È stato inoltre reimplementato il metodo propertyChange e sviluppata la sottoclasse JaudioPlayer che si occupa dell'acquisizione e riproduzione del flusso audio proveniente dalla videocamera.

4.3.2 Codice Sorgente

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.sound.sampled.*;
import de.humatic.dsj.*;
import java.io.*;

public class Frame1
    extends JFrame
    implements java.beans.PropertyChangeListener {
  JPanel contentPane;
  JPanel jPVideo = new JPanel();
  JPanel jPButton = new JPanel();
  JButton jBPause = new JButton();
  JButton jBStop = new JButton();
  JButton jBStart = new JButton();
  int check;
  String path;
  String nomeF;
  DSFiltergraph dsfg; //utilizzato per creare il flusso video
  boolean running = true;
  int w,h,renderingMode;
  JAudioPlayer jap; //utilizzato per la riproduzione del flusso audio

public static void main(String[] args){
        DSEnvironment.unlockDLL("andreamitte@katamail.com", 618634208);
      boolean packFrame = false;
      Frame1 frame = new Frame1();
    //Validate frames that have preset sizes
```

```
    //Pack frames that have useful preferred size info, e.g. from their layout
    if (packFrame) {
      frame.pack();
    }
    else {
      frame.validate();
    }
    //Center the window
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    frame.setLocation((screenSize.width - frameSize.width) / 2,
(screenSize.height - frameSize.height) / 2);
    frame.setVisible(true);}


  //Costruizione frame
  public Frame1() {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try {
      jbInit();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
   nomeFile();
    LoadDv();
  }

  //inizializzazione componenti
  private void jbInit() throws Exception {

      contentPane = (JPanel)this.getContentPane();
    contentPane.setLayout(null);
    jPVideo.setLayout(new BorderLayout());
    this.setSize(new Dimension(541, 419));
    this.setTitle("Frame Title");
    jPVideo.setBorder(BorderFactory.createEtchedBorder());
    jPVideo.setBounds(new Rectangle(3, 4, 460, 327));
    jPButton.setBorder(BorderFactory.createEtchedBorder());
    jPButton.setBounds(new Rectangle(10, 354, 454, 41));
    jBPause.setText("Pause");
    jBPause.addActionListener(new Frame1_jBPause_actionAdapter(this));
    jBStop.setText("Stop");
    jBStop.addActionListener(new Frame1_jBStop_actionAdapter(this));
    jBStart.setText("Start");
    jBStart.addActionListener(new Frame1_jBStart_actionAdapter(this));

this.setPreferredSize(new Dimension(800,600));
    contentPane.add(jPButton, null);
    jPButton.add(jBStart, null);
    jPButton.add(jBPause, null);
    jPButton.add(jBStop, null);
    contentPane.add(jPVideo, null);
    nomeF="";
    w = 300;
    h = 300;
    check=0;
```

```
  }

// permette l'inserimento del nome del file da salvare
  public void nomeFile() {
        while ( (nomeF.compareTo("") == 0)) {
      nomeF = JOptionPane.showInputDialog(this,
          "Inserire il nome del file senza estensione");
    }
  }

 //lancia il metodo per caricare il flusso proveniente dalla videocamera
  public void LoadDv() {

      loadMovie("dv", DSFiltergraph.HEADLESS);
        }


 //in base alla variabile path passata carica il grafo o cattura il flusso della
videocamera
 //collegata alla porta dv.
  private void loadMovie(String path, int flags) {
   running = false;
    if (dsfg != null) {//rimuove un oggetto dsfg creato in precedenza
      jPVideo.remove(dsfg.asComponent());
      dsfg.dispose(); //Disposes off all native & java structures
    }
    try {
      dsfg = DSFiltergraph.createDSFiltergraph(path, flags, this); //crea la
sottoclasse necessaria a svolgere l'operazione
      DSFiltergraph.DSAudioStream das = dsfg.getAudioStream(); //ottiene il
flusso audio
      if (das != null) {
        jap = new JAudioPlayer(das); //crea un oggetto JAudioPlayer per la
gestione dell'audio
      }
    }
    catch (IllegalArgumentException e) {
      System.out.println("\n" + e.toString());
      dsfg = null;
      pack();
      return;
    }
   initDisplay();
    if (jap != null) jap.start();
  }

  //metodo implementato per poter utilizzare il metodo createDSFiltergraph che
richiede come ultimo parametro
  //una variabile di tipo PropertyChangeListener
  public void propertyChange(java.beans.PropertyChangeEvent pe) {
    switch (Integer.valueOf(pe.getNewValue().toString()).intValue()) {
      case DSFiltergraph.ACTIVATING:
        System.out.print(".");
        break;
      case DSFiltergraph.DONE:
        System.out.println("done playing");
        break;
      case DSFiltergraph.LOOP:
        System.out.println("loop");
        break;
      case DSFiltergraph.EXIT_FS:
        add("Center", dsfg.asComponent());
        pack();
```

```
        break;
      case DSFiltergraph.DV_STATE_CHANGED:
       break;
      case DSFiltergraph.CAP_STATE_CHANGED:
        pack();
        break;
      case DSFiltergraph.FORMAT_CHANGED:
        w = dsfg.audioOnly ? 400 : (int) dsfg.getSize().getWidth();
        h = dsfg.audioOnly ? 10 : (int) dsfg.getSize().getHeight();
        setSize(new Dimension(w + 8, h + 72));
       pack();
        setLocation( (int) (Toolkit.getDefaultToolkit().getScreenSize().
                            getWidth() / 2 - w / 2),
                    (int) (Toolkit.getDefaultToolkit().getScreenSize().
                           getHeight() / 2 - h / 2));
        break;
      case DSFiltergraph.EXPORT_PROGRESS:
        System.out.println("% done: " +
                           Integer.valueOf(pe.getOldValue().toString()).
                           intValue());
       break;
    }
  }

  //aggiunge al JPVideo l'oggetto contenente il video
    private void initDisplay() {
    System.out.println(dsfg.getInfo());
    jPVideo.add("Center", dsfg.asComponent());
    pack();
    running = true;
    toFront();
  }

   //Riscrittura del metodo in modo tale che alla chiusura della finestra
l'applicazione termina
  protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
      running = false;
      try {
        jPVideo.remove(dsfg.asComponent());
        dsfg.dispose();
      }
      catch (Exception ex) {}

      System.exit(0);
    }
  }

  //se la variabile check a valore 0 lancia il metodo per caricare il grafo che
permette di effettuare la registrazione
  //se ha valore pari ad 1 riavvia la registrazione.
  //Bisogna ricordarsi di settare il path dei file .grf correttamente
  void jBStart_actionPerformed(ActionEvent e) {
   if (check == 0) {
      check = 1;
      path ="C:/Documents and
Settings/Andrea/Desktop/Acquisizione/Acquisisci.GRF";
      loadMovie(path, DSFiltergraph.DELIVER_AUDIO);
    }
    else dsfg.play();
  }
```

```
  //mette in pausa la registrazione
  void jBPause_actionPerformed(ActionEvent e) {
    dsfg.pause();
  }

  //stoppa la registrazione, rinomina il file intermedio temp con il nome
inserito all'inizio
  //e chiude l'applicazione
  void jBStop_actionPerformed(ActionEvent e) {
    try {
      remove(dsfg.asComponent());
      dsfg.dispose();
    }
    catch (Exception ex) {}

    // File (or directory) with old name
    File file = new File("temp.avi");
    boolean exists = file.exists();
    if (exists) {
      // File (or directory) with new name
      File file2 = new File(nomeF + ".avi");
      // Rename file (or directory)
      boolean success = file.renameTo(file2);
    }
    System.exit(0);

  }

  //classe che permette l'acquisizione e riproduzione del flusso audio
  private class JAudioPlayer
      extends Thread {
    private AudioInputStream ais;
    private AudioFormat format;
    private SourceDataLine line;
    private int bufferSize;
    private DataLine.Info info;
    private JAudioPlayer(DSFiltergraph.DSAudioStream dsAudio) {
      try {
        format = dsAudio.getFormat();
        bufferSize = dsAudio.getBufferSize();
        ais = new AudioInputStream(dsAudio, format, -1);
        info = new DataLine.Info(SourceDataLine.class, format);
        if (!AudioSystem.isLineSupported(info)) {
          System.out.println("Line matching " + info + " not supported.");
          return;
        }
      }
     catch (Exception e) {}
    }

    public void run() {
     try {
        line = (SourceDataLine) AudioSystem.getLine(info);
        line.open(format, bufferSize);
        line.start();
      }
      catch (LineUnavailableException ex) {
        System.out.println("Unable to open the line: " + ex);
        return;
      }
      try {
        byte[] data = new byte[bufferSize];
        int numBytesRead = 0;
```

```
        int written = 0;
        while (running) {
          try {
            if ( (numBytesRead = ais.read(data)) == -1)break;
            int numBytesRemaining = numBytesRead;
            while (numBytesRemaining > 0) {
              written = line.write(data, 0, numBytesRemaining);
              numBytesRemaining -= written;
            }
          }
          catch (Exception e) {
            e.printStackTrace();
            System.out.println("Error during playback: " + e);
            break;
          }
        }
        line.stop();
        line.flush();
        line.close();
      }
      catch (Exception e) {}
    }
  }
}
```

4.4 Processo di conversione dei filmati

Il filmato ottenuto dall'acquisizione ha una dimensione e un formato poco utili per la distribuzione sia su supporto fisico che via web e per questo vi è stata la necessità di codificarlo in un altro formato per ridurne le dimensioni, ma cercando di mantenere una buona qualità delle immagini.

In L.O.D.E. al termine della registrazione si ottiene un filmato in formato asf che per essere convertito in mp4, formato ottimale per la distribuzione, viene spedito al server Laurin che lo riceve, lo decodifica e successivamente lo ricodifica in mp4.

In Java per effettuare la conversione si è deciso di utilizzare FFMPEG, un convertitore audio e video potente, veloce e semplice da utilizzare. Sfruttando la capacità di modificare gran parte delle caratteristiche di un filmato si è riusciti ad ottenere un buon compromesso tra dimensione del filmato e sua qualità.

	Codifica Laurin	***Codifica Ffmpeg***
Formato input	**.asf**	**.avi**
Formato output	.mp4	.mp4
Tempo medio codifica dei filmati per ora	30 minuti	10 minuti
Bitrate (kbit/s)	264	90-190
Dimensione (frame)	360*288	360*288 - 176*144

	Codifica Laurin	***Codifica Ffmpeg***
Dimensione file iniziale (MB)	250-300	260-290
Dimensione file dopo conversione (MB)	150	70-120

Confronto processo di conversione

Osservando la tabella possiamo notare che l'impiego di Ffmpeg per effettuare la codifica in mp4 porta notevoli vantaggi. Il primo tra tutti è la riduzione del tempo di codifica che da 30 e passato a 10 minuti circa per ogni ora di filmato. Inoltre si può notare che per tempo di conversione costante è possibile scegliere un bitrate che varia da i 90 a 190 kbit/s e, nonostante sia inferiore a quello utilizzato con il vecchio metodo, può essere considerato un buon intervallo dato che la qualità del filmato rimane comunque buona e la misura 176*144 può anche essere utilizzata come dimensione in frame del filmato.

4.4.1 Realizzazione in Java

L'applicazione che realizza la conversione non ha interfaccia grafica ed è molto semplice e compatta. La prima operazione che viene eseguita è la ricerca del file con estensione avi da convertire. Per fare questo è stata creata la classe FileExtFilter, che implementa FilenameFilter, la quale passandogli l'estensione del file costruisce la stringa ".avi". È stato inoltre riscritto il metodo booleano "accept", anche se questo non verrà mai utilizzato.

Successivamente nella classe principale viene creato un oggetto File rappresentante la directory contenente il filmato da convertire, ricercato il nome del file all'interno di essa e salvato in un array di stringhe. FFMPEG può essere utilizzato solo attraverso riga di comando, quindi viene costruita una stringa contenente il comando, che andrebbe scritto nel prompt di MS-DOS e passata successivamente al metodo Runtime.getRuntime(), che permette di eseguire processi esterni alla Java Virtual Machine. L'attuale limite di questa applicazione consiste nel fatto che non si è in grado di determinare quando finisce la conversione del filmato. Per risolvere questo problema è stata costruita un'applicazione che utilizzasse un thread nel quale eseguire la conversione, ma non è stato ottenuto alcun risultato in quanto all'avvio del thread il processo di conversione partiva e si bloccava subito. Sostituendo la stringa necessaria per far partire Ffmpeg con una che, ad esempio, lanciasse il Notepad, tutto l'applicativo funzionava correttamente. Il problema non sta quindi nella costruzione dell'applicazione o nel non funzionamento del comando passato al promp di MSDos, ma nell'incapacità di far lavorare assieme Ffmpeg e Java thread.

4.4.2 Codice Sorgente

```
import java.io.*;

//effettua la conversione da avi a mp4
public class Convert extends Thread  {
      String command="";


      public static void main(String[] _args){
            System.out.print(System.getProperty("java.version"));
      new Convert();
      }

      public Convert(){
      conv();
      }

      //classe utilizzata per effettuare la ricerca del file intermedio da
convertire
      //in base all'estesione
      class FileExtFilter implements FilenameFilter
      {
          private String estensione;

          public FileExtFilter (String estensione)
          {
              this.estensione = "." + estensione;
          }

          public boolean accept (File dir, String name)
          {
              return name.endsWith (estensione);
          }
      }

      //esegue il comando dos passatogli come parametro
      public void Esegui(String command){
      try{
                  Runtime.getRuntime().exec(command);
            }catch(Exception e){System.out.println(e);}
}

//ricerca il file con estensione .avi e lo salva in una lista.
//lancia il metodo per eseguire il comando dos per la conversione
      public void conv(){

      File dir = new File ("Temp/.");
        /*-- filtro per i file .avi --*/
        FileExtFilter fef = new FileExtFilter ("avi");
        String[] list = dir.list (fef);
        String a=list[0];
          int x=a.indexOf(".");
          a=a.substring(0,x);

//ricordarsi di settare correttamente il path del file da convertire e finale
          command="cmd /C  ffmpeg.exe -i \"C:/Documents and
Settings/Andrea/Desktop/Conversione/Temp/"+a+".avi\" -b 90 \"C:/Documents and
Settings/Andrea/Desktop/Conversione/Temp/"+a+".mp4\"";
          Esegui(command);
      }     }
```

4.5 Processo di conversione presentazioni

In L.O.D.E. la conversione delle presentazioni avviene con il modulo PPT-service che grazie ad un componente Delphi permette di agganciarsi ai file .ppt. Una volta ricevuto il file il modulo scompone la presentazione e restituisce un vettore di immagini.

Un forte limite imposto da questa implementazione sta nel fatto che non è possibile utilizzare presentazioni diverse da quelle realizzate con PowerPoint e con la versione in Java si è deciso di introdurre la possibilità di utilizzare anche il formato PDF, essendo questi due i tipi di file più utilizzati per le presentazioni.

Non includendo Java una libreria per la gestione dei file Ppt e Pdf è stato necessario cercare uno strumento o un'Api esterna che potesse gestire questi tipi di file. Dopo una ricerca sul web si è deciso di utilizzare il progetto Jacob per elaborare le presentazioni Ppt e Jpedal per quelle in formato Pdf.

4.5.1 Realizzazione in Java

Al momento, per questioni di praticità nell'implementazione esistono due applicazioni, una per la conversione dei file Ppt e l'altra per la conversione dei Pdf. Nel momento in cui sarà necessario inserirle all'interno del progetto L.O.D.E. non si presenterà alcuna difficoltà nella loro unione in quanto sono state sviluppate entrambe su un'unica classe, con un contenuto numero di righe di codice e senza particolari accorgimenti per quanto riguarda la loro struttura.

Entrambe le applicazioni non presentano interfaccia grafica per il semplice motivo che l'unica operazione che l'utente deve svolgere è quella di scegliere il file da convertire, mediante un filechooser, e attendere la conversione in immagini.

Conversione presentazione PowerPoint

L'applicazione per i file PowerPoint è composta da una serie di metodi che si occupano di ottenere la presentazione, scomporla, salvarla in immagini, chiuderla e rilasciare il componente ActiveXControl. Il costruttore inizializza:

- la variabile “pptapp” richiamando il costruttore “new ActiveXComponent("PowerPoint.Application")” che crea un collegamento con le applicazioni Microsoft e più nello specifico con Powerpoint;
- la viariabile “ppto” con una copia sotto forma di oggetto della variabile “pptapp”;
- la variabile di tipo oggetto “ppts” con il metodo “Dispatch.get(Dispatch, String).toDispatch()” che passando come parametri la variabile “ppto” con un cast a

Dispatch e la stringa “Presentations” definisce il tipo di file che successivamente verrà elaborato.

Il metodo “getPresentation(String)” ricevendo come parametro una stringa contenente il path della presentazione da convetire restituisce il file sotto forma di Dispatch; per fare questo è stato utilizzato il metodo “Dispatch.call().toDispatch” passando come parametri “(Dispatch)ppts, "Open", fileName, new Variant(P.msoTrue), new Variant(P.msoTrue), new Variant(P.msoFalse)”. Il primo parametro rappresenta la presentazione ottenuta in precedenza, il secondo invece l'operazione da svolgere, il terzo il path del file e gli altri parametri rappresentano tre nuove variabili di tipo Variant inizializzate con valori booleani codificati con le specifiche Microsoft. Gli oggetti di tipo Variant rappresentano dei tipi di dato multiformato che vengono utilizzati per gran parte delle comunicazione tra Java e COM

Il metodo per salvare le singole immagini, denominato “saveAs” riceve in ingresso un oggetto di tipo Dispatch contenente la presentazione, il path dove salvare le immagini e un valore intero che indica il formato dell’immagine. I formati supportati sono molteplici e si può scegliere il più adatto alle proprie esigenze consultando la seguente tabella e modificando il valore intero passato alla funzione.

ppSaveAsAddIn	8		ppSaveAsPowerPoint3	4
ppSaveAsBMP	19		ppSaveAsPowerPoint4	3
ppSaveAsDefault	11		ppSaveAsPowerPoint4FarEast	10
ppSaveAsEMF	23		ppSaveAsPowerPoint7	2
ppSaveAsGIF	16		ppSaveAsPresentation	1
ppSaveAsHTML	12		ppSaveAsPresForReview	22
ppSaveAsHTMLDual	14		ppSaveAsRTF	6
ppSaveAsHTMLv3	13		ppSaveAsShow	7
ppSaveAsJPG	17		ppSaveAsTemplate	5
ppSaveAsMetaFile	15		ppSaveAsTIF	21
ppSaveAsPNG	18		ppSaveAsWebArchive	20

I metodi “closePresentation(Dispatch)” e “quit()” rispettivamente chiudono l'oggetto che rappresenta la presentazione PowerPoint e rilasciano il processo che connette l'applicazione ai servizi Microsoft.

Conversione presentazione Pdf

L'applicazione per la conversione dei file Pdf è costituita da due metodi principali: uno che si occupa del controllo della corretta estensione del file, della costruzione della directory di output e della chiamata del secondo metodo, che ha il compito di eseguire la conversione in immagini della presentazione in formato pdf.

Il metodo *ExtractImage(String file_name,String output_dir)* passandogli il file da convertire e la directory di output inizializzata a “null” per prima cosa controlla che la directory in cui si trova il file termini con un separatore e in caso contrario ne aggiunge uno.

Verifica poi che l'estensione del file sia “.pdf” e in caso positivo definisce la directory di output partendo da quella in cui si trova il file, aggiungendogli il nome della cartella nella quale andranno salvate le immagini e un separatore. Infine richiama il metodo per la decodifica del file.

La funzione di decodifica denominata *decodeFile(String file_name, String output_dir)* rappresenta il nucleo dell'applicazione ed accetta due parametri: il nome del file completo di percorso e la directory di output settata correttamente. La prima operazione svolta è quella di isolare dal percorso assoluto e dalla sua estensione il nome del file che verrà successivamente utilizzato per costruire il nome delle immagini che verranno create.

Viene poi inizializzato l'oggetto “PdfDecoder” che permette di decodificare una file pdf, e richiamato su di esso il metodo *setExtractionMode(int mode, int imageDpi,float scaling)* che setta l'estrazione delle immagini e il metodo *openPdfFile(String filenName)* che apre il file pdf e ottiene le informazioni chiave per poter decodificare ogni pagina. Il primo parametro passato al metodo *setExtractionMode* rappresenta come deve essere estratto il contenuto del file (immagini, testo), il secondo identifica i dpi dell'immagine e l'ultimo sono i dpi della pagina in fattore 72.

A questo punto dopo aver controllato che il file non sia protetto da password e che il contenuto sia estraibile viene creata la directory di output se non esiste, viene definito il range di pagine da convertire e comincia ad estrarre i dati dal pdf elaborando una pagina per volta.

Viene costruito il nome dell'immagine che sarà salvata utilizzando la stringa isolata in precedenza più il numero della pagina che si sta estraendo, successivamente è creato un oggetto “BufferedImage” e inizializzato utilizzando il metodo *getPageAsImage(int pageIndex)* richiamato sempre sull'oggetto PdfDecoder. L'applicazione offre anche la possibilità di creare immagini con una dimensione ridotta. Di default vengono create immagini aventi la stessa dimensione della

pagina contenuta nel file pdf, ma cambiando il valore della variabile "*scaling"* possiamo ridurre o aumentare le dimensioni.

Per fare questo viene ricalcolata la grandezza della pagina in base al valore inserito, creato un oggetto Image e inizializzato applicando all'oggetto "BufferedImage" il metodo *getScaledInstance(int width, int height, int hints)* che ridimensiona l'immagine originale. I parametri passati sono la larghezza, l'altezza e il bit rappresentante l'algoritmo da usare per il ridimensionamento, in questo caso è stato utilizzato *BufferedImage.SCALE_SMOOTH* che dà maggiore priorità alla qualità dell'immagine e alla velocità di esecuzione dell'operazione.

Altri flag che identificano un tipo di algoritmo sono:

- SCALE_DEFAULT: utilizza l'algoritmo di default per il ridimensionamento delle immagini.
- SCALE_FAST: sceglie un algoritmo che predilige la velocità di esecuzione dell'operazione rispetto alla qualità dell'immagine.
- SCALE_REPLICATE: utilizza l'algoritmo incluso nella classe *ReplicateScaleFilter*.
- SCALE_AREA_AVERAGING: utilizza l'algoritmo di ridimensionamento Area Average.

A questo punto l'oggetto "BufferedImage" viene inizializzato richiamando il costruttore *BufferedImage(int width, int height, int imageType)*. I parametri rappresentano la larghezza, l'altezza e il tipo di immagine creata.

Quest'ultimo parametro ha valore *BufferedImage.TYPE_INT_RGB* e rappresenta un immagine con colori RGB a 8 bit racchiusi in pixel interi. Viene creato ora un oggetto "Graphics2D", inizializzato con il metodo *createGraphics()* applicato sull'oggetto "BufferedImage" e disegnata l'immagine ridimensionata con il metodo *drawImage(scaledImage, 0, 0 ,null)* sul "Graphics2D" appena creato.

Ultima operazione svolta sia che si tratti di una pagina a dimensioni normali che ridimensionata è il salvataggio dell'immagine su file richiamando sull'oggetto PdfDecoder il metodo *getObjectStore().saveStoredImage(String current_image, BufferedImage image, boolean file_name_is_path, boolean save_unclipped,String type).*

Nel caso in cui l'operazione di estrazione e salvataggio delle pagine ha esito positivo il file pdf viene chiuso e la procedura è terminata.

L'applicazione così implementata non presentata deficit nella sua struttura e funzionamento.

4.5.2 Codici Sorgente

Conversione presentazioni in formato Pdf

```
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Iterator;
import java.util.List;
import javax.swing.*;
import org.apache.log4j.Level;
import org.pdfbox.exceptions.COSVisitorException;
import org.pdfbox.pdmodel.PDDocument;
import org.pdfbox.pdmodel.PDPage;
import com.sun.image.codec.jpeg.JPEGCodec;
import com.sun.image.codec.jpeg.JPEGImageEncoder;

public class ConvertPdf {
    public ConvertPdf() {
        super();
    }
    public static void main(final String[] args) throws COSVisitorException,
            IOException {
        ConvertPdf me = new ConvertPdf();
        JFileChooser c=new JFileChooser();
        c.showDialog(null,"Seleziona il file da convertire");
        me.convertPDFtoJPEG((c.getSelectedFile()));
        JOptionPane.showMessageDialog(null, "Conversione avvenuta");
       }

    public void convertPDFtoJPEG(final File target) throws IOException {
      org.apache.log4j.BasicConfigurator.configure();
      org.apache.log4j.Logger.getRootLogger().setLevel(Level.INFO);
              PDDocument document = null;
        try {
            document = PDDocument.load(target);
            List pages = document.getDocumentCatalog().getAllPages();
            int count = 0;
            for (Iterator i = pages.iterator(); i.hasNext(); ) {
                  PDPage page = (PDPage)i.next();
                BufferedImage image = page.convertToImage();
                FileOutputStream fos = new FileOutputStream(target
                        .getAbsolutePath()
                        + "." + count++ + ".jpg");
                JPEGImageEncoder encoder = JPEGCodec.createJPEGEncoder(fos);
                encoder.encode(image);
                fos.close();
            }
        } finally {
            if (document != null) {
                document.close();
            }
        }
    }
}
```

Conversione presentazioni in formato PowerPoint

```
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
import com.jacob.activeX.*;
import com.jacob.com.*;

public class Ppt {
      ActiveXComponent pptapp = null;  //PowerPoint.Application ActiveXControl
Object
        Object ppto = null;  //PowerPoint.Application COM Automation Object
        Object ppts = null;  //Presentations Set
        // Office.MsoTriState
        public static final int msoTrue = -1;
        public static final int msoFalse = 0;
        // PpSaveAsFileType
        public static final int ppSaveAsJPG = 17 ;


        public Ppt(){
              try{
                  pptapp = new ActiveXComponent("PowerPoint.Application");
                  ppto = pptapp.getObject();
                  ppts = Dispatch.get((Dispatch)ppto,
"Presentations").toDispatch();
                }catch(Exception e){
                  e.printStackTrace();
                }
              }

              public Dispatch getPresentation(String fileName){
                Dispatch pres = null; //Presentation Object
                try{
                  pres = Dispatch.call((Dispatch)ppts, "Open", fileName,
                    new Variant(Ppt.msoTrue), new Variant(Ppt.msoTrue),
                    new Variant(Ppt.msoFalse)).toDispatch();
                }catch(Exception e){
                  e.printStackTrace();
                }
                return pres;
              }
              public void  saveAs(Dispatch presentation, String saveTo, int
ppSaveAsFileType){
                try{
                  Object slides = Dispatch.get(presentation,
"Slides").toDispatch();
                  Dispatch.call(presentation, "SaveAs", saveTo, new
Variant(ppSaveAsFileType));
                }catch (Exception e) {
                  e.printStackTrace();
                }
              }
              public void closePresentation(Dispatch presentation){
                  if(presentation != null){
                  Dispatch.call(presentation, "Close");
                }
              }
              public void quit(){
                  if(pptapp != null){
                  ComThread.Release();
                  //pptapp.release();
                  pptapp.invoke("Quit", new Variant[]{});
```

```
                }
              }
        public static void main(String[] args){
            Ppt a = new Ppt();
            String p="";
            JFileChooser c=new JFileChooser();
            c.showDialog(null,"Seleziona il file da convertire");
            System.out.println(c.getSelectedFile().getAbsolutePath());
            p=c.getSelectedFile().getAbsolutePath();
            String out = p.replaceAll ("\\\\", "\\\\\\\\");
            System.out.println(out);
        //  Dispatch pres = a.getPresentation(out);//inserire percorso file ppt
            Dispatch pres = a.getPresentation(p);//inserire percorso file ppt
            a.saveAs(pres, "C:\\Documents and
Settings\\Andrea\\Desktop\\progetti\\PptToJpeg\\SlidesJpg", Ppt.ppSaveAsJPG);
//inserire percorso destinazione file
          a.closePresentation(pres);
          a.quit();
          JOptionPane.showMessageDialog(null, "Conversione avvenuta");
          System.out.println("fine");
        }
}
```

5. CONCLUSIONI

5.1 Difficoltà incontrate

Questa sezione tratta le principali difficoltà incontrate nella ricerca, nello studio del materiale utilizzato e nella realizzazione degli applicativi presentati. Nell'elaborazione concettuale della struttura delle applicazioni non sono stati riscontrati particolari problemi in quanto si è potuto prendere spunto da ciò che era stato pensato e sviluppato nel progetto L.O.D.E. apportando le opportune modifiche, necessarie per l'integrazione con l'ambiente Java e gli strumenti software che si è deciso di utilizzare. La ricerca dei tools adatti ha portato qualche inconveniente perché la scarsa documentazione ha reso un po' difficoltosa la raccolta delle informazioni necessarie per decidere se si trattava dello strumento adatto a svolgere le operazioni richieste.

Per quanto riguarda l'implementazione in Java delle applicazioni grazie agli insegnamenti impartiti durante le lezioni dei corsi e l'esperienza personale non ho trovato difficoltà rilevanti.

6. BIBLIOGRAFIA

[1] Dolzani M., relatore Ronchetti M. “L.O.D.E.: un sistema per la produzione e la fruizione di lezioni multimediali”. Tesi di laurea in Informatica, Università degli studi di Trento, Facoltà di Scienze Matematiche, Fisiche, Naturali. Anno accademico 2003/2004.
[2] Dolzani M., Ronchetti M., “Video Streaming over the Internet to support learning: the L.O.D.E. system”. *WIT Transactions on Informatics and Communication Technologies*, 2005, v. 34, p. 61-65.

Dolzani M., Ronchetti M. (2005). Lectures On DEmand: the architecture of a system for delivering traditional lectures over the Web. In Kommers, P., & Richards, G. (Eds.), *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2005* (pp. 1702-1709). Chesapeake, VA: AACE.

Ronchetti M., “Has the time come for using video-based lectures over the Internet? A Test-case report” "*CATE - Web Based Education Conference 2003*", Rhodes (Greece), June 30 - July 2, 2003

Ronchetti, M. (2003). Using the Web for diffusing multimedia lectures: a case study. In Kommers, P., & Richards, G. (Eds.), *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2003* (pp. 337-340). Chesapeake, VA: AACE.
[3] http://www.humatic.de/htools/dsj.htm
[4] http://www.humatic.de/htools/dsj/javadoc/index.html
[5] http://ffmpeg.mplayerhq.hu/ffmpeg-doc.html
[6] http://soenkerohde.com/tutorials/ffmpeg/
[7] http://windowssdk.msdn.microsoft.com/en-us/library/ms787656.aspx
[8] http://www.jpedal.org/
[9] http://danadler.com/jacob/
[10] http://java.sun.com/products/java-media/jmf/
[11] https://sourceforge.net/forum/forum.php?thread_id=1472161&forum_id=116813
[12] http://it.wikipedia.org/wiki/Digital_Video
[13] http://it.wikipedia.org/wiki/FireWire
[14] http://it.wikipedia.org/wiki/XviD
[15] http://it.wikipedia.org/wiki/Audio_Video_Interleave
[16] http://www.microsoft.com/com/default.mspx
[17] http://it.wikipedia.org/wiki/Dolby_Digital