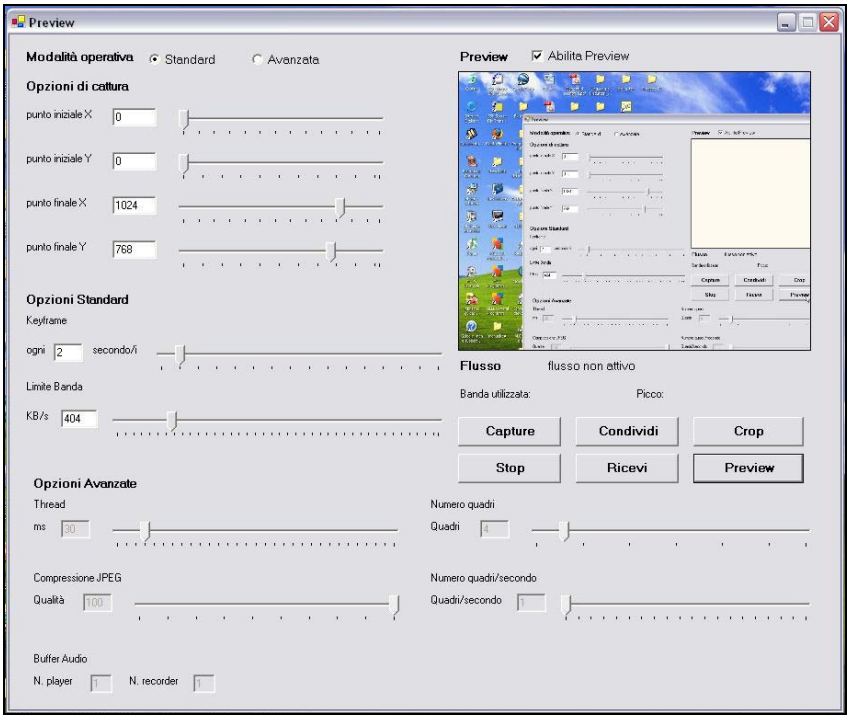
DesktopSharing

Condivisione di audio e video

Febbraio 2006
Marco Vallini
marcovallini@gmail.com



Obiettivi
Si realizzi una applicazione che permetta la condivisione in rete del contenuto dello schermo di un calcolatore e la diffusione di un commento audio associato.

L'applicazione permette la trasmissione in streaming del contenuto dello schermo presente sul calcolatore e del segnale audio proveniente dalla scheda sonora ad un insieme di destinatari, utilizzando la tecnica IP multicast.

L'applicazione sviluppata potrà operare sia come destinatario che come sorgente.

In modalità sorgente, l'applicazione deve consentire di:

- indicare quale porzione dello schermo deve essere trasmessa
- indicare il bitrate massimo da utilizzare per la trasmissione dei dati
- selezionare un oppportuno indirizzo IP multicast
- attivare e sospendere la trasmissione del flusso video all'indirizzo selezionato

In modalità destinatario, l'applicazione deve consentire di:

- indicare l'indirizzo multicast per la ricezione del flusso video
- ricostruire e visualizzare il contenuto dello schermo del server
- ricostruire e riprodurre il segnale audio

Requisiti
Il sistema deve permettere la trasmissione di due flussi di informazione limitando il consumo di risorse condivise e massimizzando la probabilità di una corretta interpretazione delle informazioni da parte dei destinatari. A questo scopo occorre che la banda complessivamente impegnata dalla sorgente non superi un limite fissato dall'utente; tale limite dovrà essere ovviamente compatibile con i requisiti minimi necessari a trasportare il flusso audio della sessione. Tale flusso dovrà, infatti, avere la priorità su quello video al fine di mantenere l'intellegibilità del contenuto.

Poiché una parte significativa della comunicazione può essere veicolata attraverso il movimento del mouse, le informazioni relative alla sua posizione dovranno essere riportate in ciascun pacchetto audio, al fine di garantirne una tempestiva visualizzazione da parte dei destinatari.

Per ridurre il consumo di banda da parte della componente video, è possibile adottare qualche forma di compressione. Data la limitata variabilità del contenuto dello schermo durante una presentazione, è possibile ottenere un primo livello di compressione suddividendo l'immagine da trasmettere in blocchi elementari ed inviando solo i blocchi nei quali si è verificata una variazione rispetto al fotogramma precendente.

Poiché il trasporto multicast è intrinsecamente inaffidabile, occorre che l'applicazione in modalità sorgente provveda ad inviare, con una certa ridondanza, le informazioni relative al video: questo può essere ottenuto mediante il concetto di "key frame"; ovvero, trascorso un certo intervallo di tempo, l'applicazione provvede a ritrasmettere tutti i sottoblocchi, anche se non sono variati. L'intervallo di tempo tra due key frame deve essere configurabile dall'utente.

Il protocollo applicativo utilizzato per la comunicazione tra i calcolatori deve essere definito in maniera opportuna in fase di progetto e può ispirarsi a protocolli esistenti per il trasferimento di contenuti multimediali (RTP).

1. Introduzione

Il progetto, che prevede la trasmissione dei flussi audio e video della parte di uno schermo di un calcolatore si divide in due componenti: trasmettitore e ricevitore, entrambi integrati nella stessa applicazione.

La comunicazione, deve sfruttare la tecnica del multicast, in cui, un elemento trasmette e gli altri ricevono. Il trasmettitore deve costituire un gruppo multicast, ed i ricevitori devono diventare membri associandosi, previa conoscenza dell'indirizzo utilizzato.

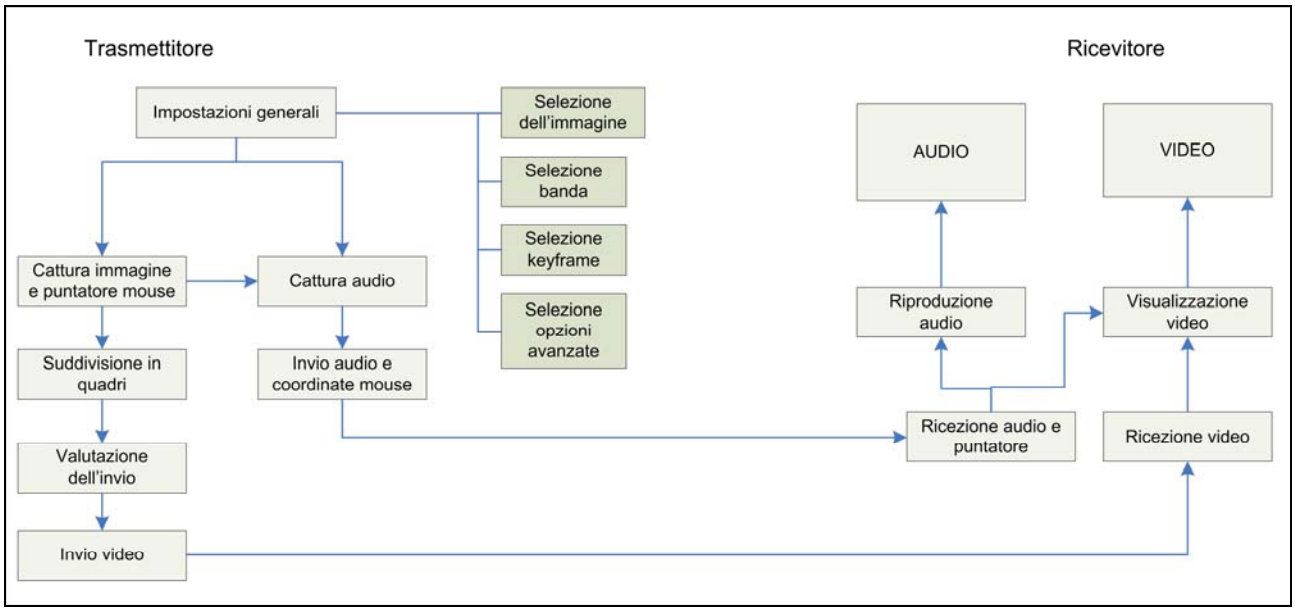
La tipologia di pacchetto utilizzato è UDP, non connesso e non affidabile. Questa scelta, permette di inviare un flusso verso un indirizzo e, la ricezione senza che sia necessario instaurare una connessione diretta tra entità client e server. Inoltre, l'overhead del protocollo è minore rispetto a TCP perché non vi sono controlli sulla consegna del pacchetto. La natura dell'applicazione, permette la perdita di alcuni pacchetti, che, se limitata con meccanismi opportuni, non agisce in modo significativo sulle prestazioni. Tuttavia, non presenta meccanismi di limitazione della banda utilizzata, rendendone necessaria l'implementazione per non saturare la rete.

Le specifiche, impongono inoltre, che sia il flusso audio che il movimento del mouse, abbiano una priorità maggiore rispetto a quello video. Ciò impone:

- suddividere i flussi audio/mouse e video, gestendoli con moduli separati
- eventualmente, per limitare la banda e consentire al flusso audio di arrivare correttamente a destinazione, di agire solo sulla qualità video

Per ottimizzare l'utilizzo delle prestazioni, ed evitare blocchi dell'applicazione, è necessaria la programmazione multithreading. Questa scelta è necessaria anche per garantire la separazione tra i flussi diversi. La famiglia dei sistemi operativi Windows 2000/2003 ed XP, e l'ambiente di programmazione .NET supportano completamente questa tecnica.

2. Architettura generale

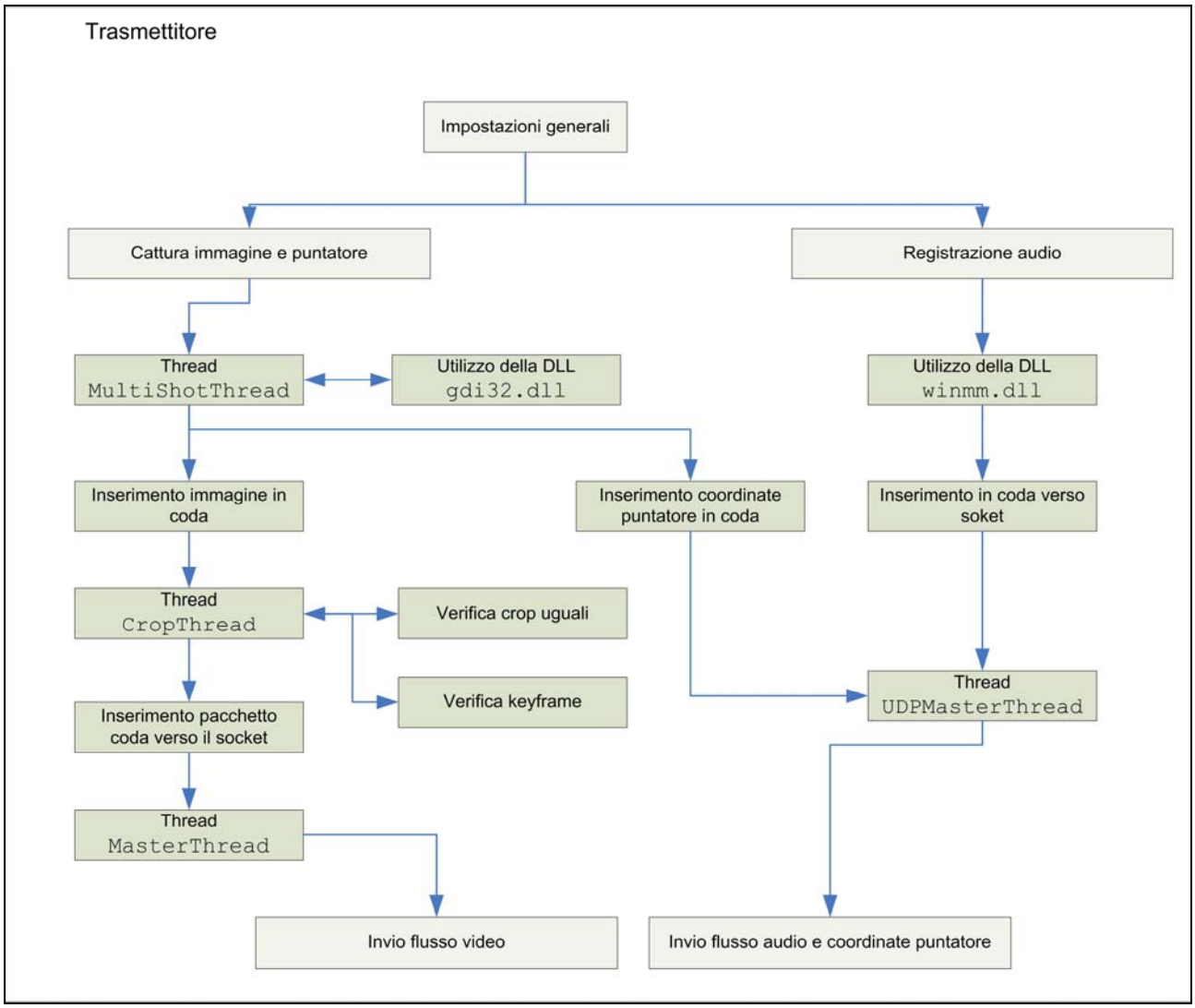


Dallo schema proposto, si osservano i blocchi logici che costituiscono trasmettitore e ricevitore. Le impostazioni generali, prevedono la selezione della dimensione della banda, del keyframe, delle dimensioni dell'immagine e delle opzioni avanzate. In queste, sono presenti: la qualità dell'immagine, il numero di frame al secondo, il numero di suddivisioni dell'immagine ed il numero di buffer da utilizzare nella registrazione e riproduzione dell'audio.

La cattura dell'immagine e la cattura del video, avvengono in parallelo, sfruttando le tecnica del multithreading. Successivamente, si avrà la suddivisione in quadri dell'immagine, si determinerà quali quadri inviare e si procederà all'invio attraverso un socket. Per quanto riguarda l'audio, dopo essere catturato sarà subito trasmesso sfruttando un socket.

Il ricevitore, ha il compito di ricevere i pacchetti dei due flussi e riprodurli. In particolare, dato che il pacchetto audio contiene anche le coordinate del video, le procedure dovranno interagire.

3. Architettura del trasmettitore



3.1 Impostazioni generali

Le impostazioni generali sono suddivise in standard e avanzate. Quelle standard sono:

- selezione dell'immagine: è possibile, mediante uno strumento grafico selezionare l'area di schermo da catturare
- keyframe: indica ogni quanti secondi viene ritrasmessa l'intera immagine
- limite banda trasmissiva: indica il limite medio della banda utilizzata, la valutazione è effettuata considerando dimensioni dell'immagine, banda audio, numero di screenshot, e numero di quadri

Quelle avanzate sono:

- Thread Time: utilizzata per controllare la tempistica del Thread che si occupa di inviare il flusso video
- Compressione JPEG: è possibile selezionare la qualità dell'immagine da inviare, che interagisce direttamente sul limite della banda trasmissiva
- Buffer Audio: è possibile specificare il numero di buffer audio sia in registrazione che in riproduzione, per consentire di eliminare pause tra il parlato
- Numero suddivisione dell'immagine: è possibile impostare il numero di quadri in cui suddividere l'immagine
- Numero di screenshot al secondo: identifica il frame-rate del video

3.2 Cattura dell'immagine e delle coordinate puntatore

La cattura dell'immagine e delle coordinate del mouse, avvengono attraverso il thread `MultiShotThread`. Questo, appoggiandosi alla classe `capture_screen`, esegue la cattura dell'immagine e, utilizzando la classe `mousePointer`, cattura le coordinate del puntatore del mouse. La classe `capture_screen` si appoggia alla DLL `gdi32`, importata all'interno dell'ambiente .NET.

Successivamente, il thread, inserisce l'immagine nella coda verso il thread `CropThread` che la suddividerà. Dato che ogni quadro, è prelevato in diverse regioni dell'immagine, il pacchetto inserito in coda, conterrà anche le informazioni sulla regione di origine del crop. Inoltre, inserisce le coordinate del puntatore nella coda condivisa con il thread `UDPMasterThread`.

Il thread `CropThread`, che è implementato nella classe `imageMgr`, riceve dalla coda l'immagine catturata e provvede a due operazioni: (1) verifica se è necessario inviare un keyframe, (2) invia solo la differenza rispetto all'immagine precedente. Nella prima, suddivide l'immagine in quadri, comprime ogni quadro con la qualità JPEG specificata e, li inserisce in una coda condivisa con il thread `MasterThread`. Nella seconda, suddivide sia l'immagine precedente che quella attuale in quadri, confronta i quadri con il metodo `CropEquals` e se sono uguali, non lo invia, altrimenti, lo comprime e lo invia nella coda condivisa. Prima dell'inserimento nella coda, i quadri sono convertiti in bytes. Il metodo `CropEquals`, esegue il confronto byte per byte.

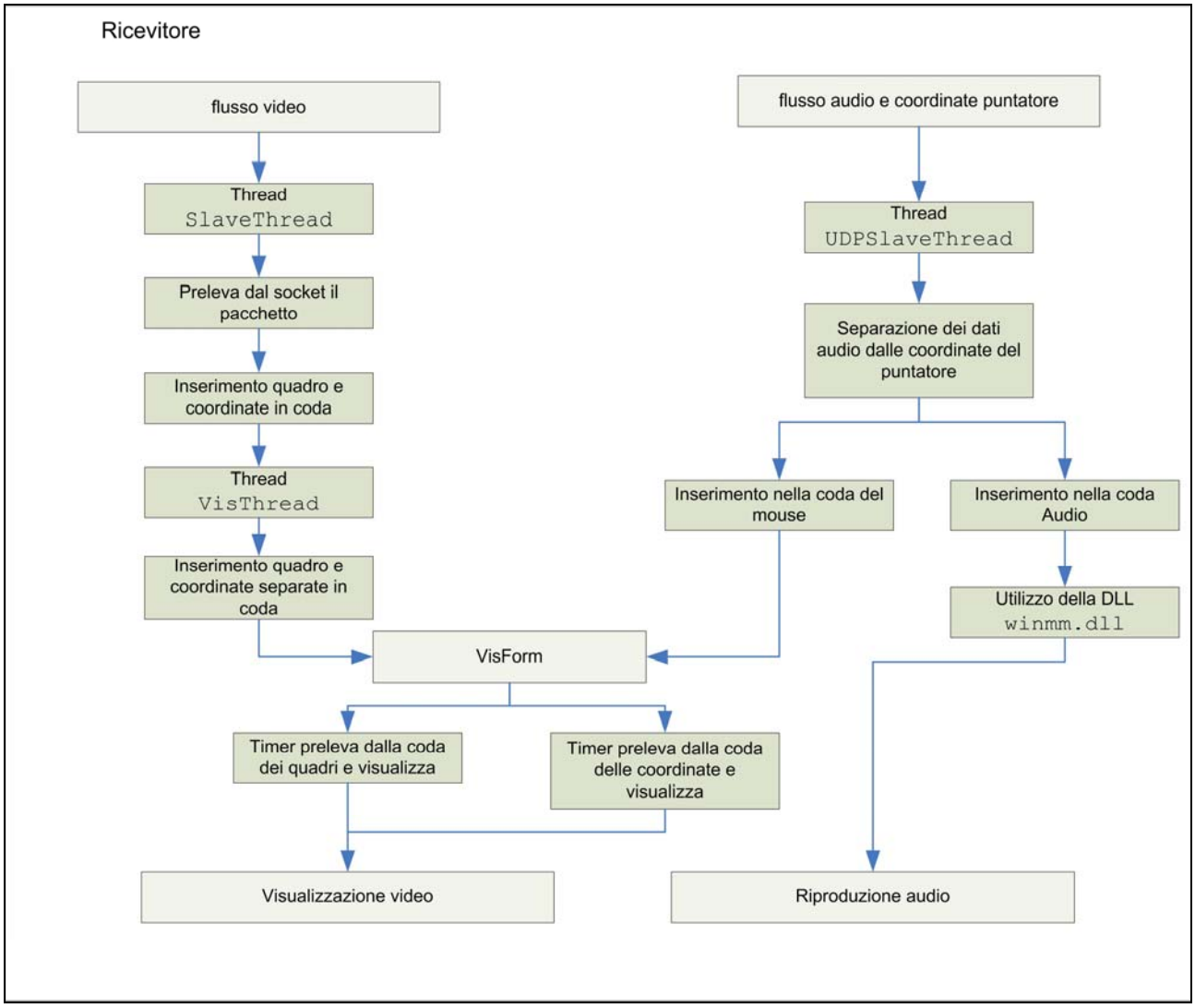
Il thread `MasterThread`, si occupa di prelevare i crop, ovvero i quadri e di imbustarli in un pacchetto UDP e di inviarli sul socket. Inoltre, nella fase di inizializzazione, registra un indirizzo multicast a cui inviare i pacchetti, specificando anche la porta del flusso video.

3.3 Registrazione dell'audio

La registrazione dell'audio avviene attraverso le classi `WaveIn` e `WaveNative`, che sfruttano la DLL `winmm.dll`, libreria multimediale di Win32, importata all'interno dell'ambiente .NET. In particolare, nella classe `WaveNative` vengono importate le chiamate alle funzioni della libreria e vengono impostati alcuni parametri, come il numero di sample, il tempo in millisecondi, offset temporale in byte, ecc. L'altra classe, si occupa invece di implementare i metodi, in particolare per la gestione dei buffer audio. La classe `audioComponent`, implementa i metodi di alto livello. Attraverso introduzione di un "eventHandler", in questa classe, è possibile richiamare il metodo `DataArrivedMaster`, che inserisce il buffer nella coda condivisa con il thread `UDPMasterThread`. In particolare, il metodo `DataArrivedMaster`, viene richiamato ogni volta che un buffer audio registrato è disponibile ad essere inviato.

Il thread `UDPMasterThread`, della classe `UDPAudioMaster`, esegue due operazioni: (1) estrarre da una coda le coordinate del mouse, (2) estrarre dall'altra coda il buffer audio. La prima operazione, dopo l'estrazione, converte le coordinate in byte. Dopo le operazioni citate, il thread, costruisce un pacchetto che contiene: (1) numero di sequenza, (2) coordinate mouse e (3) buffer audio registrato. Successivamente, invia il pacchetto così formato sul socket. Il numero di sequenza, è utile per il pacchetto audio in quanto, se i diversi pacchetti, fossero instradati in modo differente, un pacchetto con numero di sequenza minore, potrebbe arrivare dopo il numero di sequenza attualmente in riproduzione. In questo caso, sarà soppresso. Inoltre, nella fase di inizializzazione, registra un indirizzo multicast a cui inviare i pacchetti, specificando anche la porta del flusso audio.

4. Architettura del ricevitore



4.1 Flusso Video

Il flusso video, viene ricevuto attraverso un socket sul quale è impostato un timeout. La ricezione avviene attraverso il meccanismo dell'indirizzo multicast, quindi, il thread, prima di poter comunicare, deve diventare membro del gruppo. Il meccanismo del timeout, è utile in quanto se il flusso dal trasmettitore verso il ricevitore dovesse interrompersi, il thread non rimarrebbe bloccato. In questo caso terminerà se non riceverà pacchetti entro cinque secondi. Il thread che implementa queste funzionalità è `SlaveThread`. In particolare, dopo aver ricevuto il pacchetto che contiene un quadro e le coordinate, lo inserisce nella coda condivisa con il thread `VisThread`.

Il thread `VisThread`, preleva dalla coda le due coordinate ed il quadro (poste in sequenza) ed invoca il metodo `fillImage` che provvede a inserire coordinate e quadro nella coda condivisa con la classe `VisForm`.

4.1.1 VisForm – aspetto del video

La classe `VisForm`, implementa i metodi per visualizzare sia l'immagine che il puntatore del mouse. Estrae dalla coda i quadri inviati, attraverso un timer, forza il refresh dell'area client che provvederà a disegnare il quadro nella regione corretta (attraverso l'utilizzo delle coordinate del crop).

4.2 Flusso Audio

Il flusso audio, viene ricevuto, come il flusso video, attraverso un socket sul quale è impostato un timeout, ed il comportamento è, sotto questo aspetto, identico al precedente. Il thread che implementa queste funzionalità è `UDPSlaveThread`. Dopo aver ricevuto il pacchetto, che contiene numero di sequenza, coordinate mouse e buffer audio, lo scompone. Innanzi tutto verifica il numero di sequenza ed eventualmente aggiorna quello attuale. Come seconda operazione, scompone il pacchetto in due: una pacchetto di coordinate e uno per il buffer audio. Il pacchetto delle coordinate, viene convertito in due numeri, ed entrambi sono inseriti nella coda del mouse condivisa con `VisForm`. Invece, il pacchetto audio viene inserito nella coda condivisa con la classe `audioComponent`.

4.2.1 VisForm – aspetto dell'audio

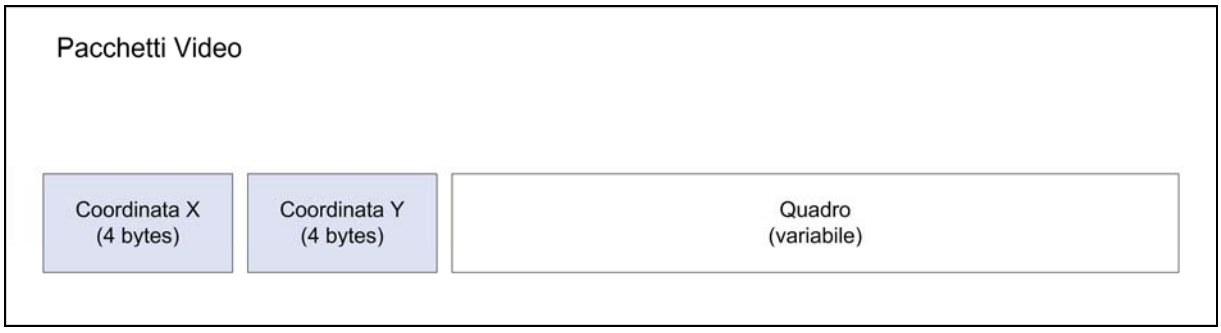
La classe `VisForm`, oltre che implementare i metodi per il disegno dell'immagine, implementa anche quelli per il puntatore del mouse. In particolare, le operazioni sono: estrazione dalla coda delle coordinate, attraverso un Timer (diverso dal precedente) che forza il refresh dell'area client, provvedendo a disegnare il puntatore alle coordinate corrette.

Per quanto riguarda il pacchetto audio, esso viene estratto dalla coda della classe `audioComponent`, quando viene generato un "eventHandler", innescato dal player audio. L'evento, provoca l'invocazione del metodo `FillerSlave` che implementa l'estrazione dalla coda e la copia del buffer audio all'interno di un'area di memoria di scambio. A questo punto, il player bufferizza l'audio e lo riproduce.

5. Struttura dei pacchetti

Vi sono due tipologie di pacchetti, quelli del flusso audio e quelli del flusso video. Riassunti in modo schematico di seguito.

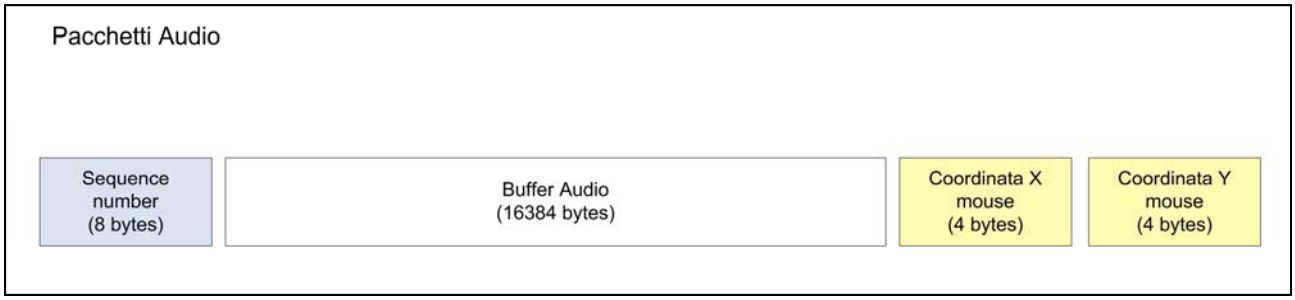
5.1 Pacchetti Video



Come si può osservare dallo schema, i pacchetti video contengono 2 campi da 4 bytes ciascuno per le coordinate del mouse, e una parte variabile che contiene il quadro. Non è possibile stabilire a priori la dimensione di questo campo, perché dipende dal numero di suddivisioni e dall'area dello

schermo catturata. Le coordinate X e Y, sono convertite da numero intero a bytes dal trasmettitore e viceversa dal ricevitore.

5.2 Pacchetti Audio



Come si può osservare dallo schema, i pacchetti audio contengono in totale 4 campi: il primo esprime il sequence number su 8 bytes, il secondo contiene il buffer audio in 16Kb e gli altri due contengono le coordinate del mouse in 4 byte ciascuno. Il sequence number è convertito da intero lungo a byte dal trasmettitore e viceversa dal ricevitore. Le coordinate, sono convertite da intero a byte dal trasmettitore e viceversa dal ricevitore.

6. Considerazioni sul multithreading

Il progetto, come detto in precedenza, è stato suddiviso in classi che utilizzano la tecnica multithreading, ovvero l'esecuzione di thread paralleli. Tuttavia, per la corretta gestione dei thread, soprattutto durante la fase di terminazione è necessario prestare attenzione. In particolare, la terminazione avviene con la chiamata del metodo `Abort()`. Questa, scatena all'interno del thread, una eccezione che deve essere gestita e resettata mediante il metodo `ResetAbort()`. Quindi, i metodi utilizzati per terminare i thread, dapprima impostano un flag che permette la sospensione delle operazioni e successivamente, chiamano il metodo `Abort()`. Successivamente chiamano il metodo `Join()` che permette di bloccare le chiamate al thread, finchè non terminerà.

7. Considerazioni sui metodi di condivisione

Per la condivisione delle informazioni sono stati utilizzati due tecniche: (1) socket sincroni, per condivisione delle informazioni tra trasmettitore e ricevitore, (2) queue, code di tipologia FIFO. La seconda tecnica è stata scelta per lo scambio di informazioni tra i thread e le classi. L'oggetto queue, così come altri oggetti per scambio dati, non può essere letta e scritta contemporaneamente, bisogna cioè utilizzare un meccanismo di sincronizzazione che permetta la gestione in mutua esclusione. Quello scelto per le code è `lock()`, che consente di specificare l'oggetto su cui eseguire la sincronizzazione, effettuare le operazioni di lettura e scrittura e rilasciare correttamente la regione critica.

8. Modalità trasmettitore

In modalità trasmettitore, l'applicazione esegue le seguenti operazioni:

1. acquisisce le impostazioni selezionate dall'utente: sia quelle standard che avanzate
2. definisce e costruisce le code condivise tra i thread e le classi: `queueFromMStoIMgr`, `queueFromIMgrtoUDP`, `queueMousePointer`
3. istanzia e configura la classe `multishotMgr`: che si occupa di richidere le catture
4. istanzia e configura la classe `imageMgr`: che si occupa di gestire crop e keyframe
5. istanzia e configura la classe `UDPMaster`: che si occupa di costruire il socket, registrare il gruppo multicast ed inviare i pacchetti video
6. istanzia e configura la classe dei componenti audio `audioComponent`
7. istanzia e configura la classe `UDPAudioMaster`: che si occupa di costruire il socket, registrare il gruppo multicast ed inviare i pacchetti audio e coordinate del mouse

8.1 Classe `multishotMgr`

La classe, si occupa attraverso la creazione del thread `MultiShotThread`, di catturare le immagini attraverso la classe `capture_screen` in base alla tempistica scelta dall'utente. Questa, è scelta selezionando il numero di screenshot al secondo. L'area di schermo da catturare, è quella selezionata dall'utente. Inoltre, viene registrata la posizione del puntatore del mouse attraverso la classe `mousePointer`. Infine queste informazioni sono inserite rispettivamente nelle code `queue` e `mouseQueue`.

8.2 Classe `imageMgr`

La classe, si occupa attraverso la creazione del thread `CropThread`, di:

1. verificare se è necessario trasmettere un keyframe
2. suddividere l'immagine in quadri a seconda del numero selezionato dall'utente
3. verificare se sono uguali
4. comprimerli a seconda della qualità stabilita dall'utente
5. registrare le coordinate del singolo quadro
6. trasformare le informazioni in bytes
7. inviare le informazioni nella coda condivisa con la classe UDPMaster

Per verificare se i quadri sono uguali utilizza il metodo `CropEquals`.

8.3 Classe `UDPMaster`

La classe, si occupa di costuire e attivare il thread `SendTh`, che esegue le seguenti operazioni:

1. costruire un socket per i pacchetti video, registrando un indirizzo multicast e una porta, entrambe passate come parametri dall'utente
2. estrarre i pacchetti dalla coda, ricevuti dal thread `CropThread`
3. inviare i dati sul socket

8.4 Classe `audioComponent` modalità trasmettitore

La classe, nella modalità trasmettitore, si occupa di:

1. istanziare un formato audio opportuno
2. instanziare un recorder specificando il metodo da richiamare dall'event handler
3. utilizzare il metodo `DataArrivedMaster` richiamato tramite event handler
4. inserire i buffer audio nella coda condivisa con la classe `UDPAudioMaster`

Inoltre, questa sfrutta i metodi implementati dalle classi `WaveIn` e `WaveNative`.

8.5 Classe `UDPAudioMaster`

La classe, si occupa di costruire e attivare il thread `UDPMasterThread`, che esegue le seguenti operazioni:

1. costruire un socket per i pacchetti audio e coordinate mouse, registrare un indirizzo multicast ed una porta, entrambe passate come parametri dall'utente
2. estrarre i pacchetti audio dalla coda condivisa con `audioComponent`
3. estrarre i pacchetti delle coordinate del mouse dalla coda condivisa con `MultiShotThread`
4. costruire un numero di sequenza
5. convertire in bytes sia le coordinate del mouse che il numero di sequenza
6. inviare le informazioni sul socket

9. Modalità ricevitore

In modalità ricevitore, l'applicazione esegue le seguenti operazioni:

1. acquisisce le impostazioni selezionate dall'utente: sia quelle standard che avanzate
2. definisce e costruisce le code condivise tra i thread e le classi: `mouseQueue, UDPToVisqueue, VisToFormqueue`
3. istanzia e configura la classe `UDPSlave`: che si occupa di costruire il socket, registrarsi al gruppo multicast e ricevere i pacchetti video
4. istanzia e configura la classe `visualizer`: che si occupa di prelevare coordinate e crop e inserirli nella coda della classe "VisForm" perché siano visualizzati
5. istanzia e configura la classe `VisForm`: che visualizza i quadri
6. istanzia e configura la classe dei componenti audio `audioComponent`
7. istanzia e configura la classe `UDPAudioSlave`: che si occupa di costruire il socket, registrarsi al gruppo multicast e ricevere i pacchetti audio e coordinate del mouse

9.1 Classe `UDPSlave`

La classe, si occupa di costruire e attivare il thread `SlaveThread`, che esegue le seguenti operazioni:

1. costruisce un socket per ricevere i pacchetti video e si registra come membro all'indirizzo multicast, passato come parametro dall'utente
2. riceve il pacchetto
3. inoltra il pacchetto nella coda condivisa con la classe `visualizer`

9.2 Classe `visualizer`

La classe, si occupa di costruire e attivare il thread `VisThread`, che esegue le seguenti operazioni:

1. preleva dalla coda condivisa con `SlaveThread` il pacchetto del quadro video e le coordinate
2. trasforma le coordinate in due numeri interi
3. trasforma il quadro in una Bitmap
4. invia coordinate e quadro nella coda condivisa con la classe `VisForm`

9.3 Classe `VisForm`

La classe, si occupa di visualizzare le informazioni in un apposito form. Queste sono: il puntatore del mouse e i quadri video. Questa presenta due timer, uno per prelevare dalla coda le coordinate del mouse, l'altro per prelevare dalla coda i quadri e le sue coordinate da visualizzare. La prima coda è condivisa con `UDPAudioSlave`, l'altra con `visualizer`. Ogni volta che i timer scattano, prelevano le informazioni dalle relative code e forzano il refresh del form. Il metodo `VisForm_Paint` consente di visualizzare quadri e puntatore.

9.4 Classe `audioComponent` modalità ricevitore

La classe, nella modalità ricevitore, si occupa di:

1. istanziare un formato audio opportuno
2. instanziare un player specificando il metodo da richiamare dall'event handler
3. utilizzare il metodo `FillerSlave` richiamato tramite event handler
4. prelevare i buffer audio dalla coda condivisa con la classe `UDPAudioSlave`

Inoltre, questa sfrutta i metodi implementati dalle classi `WaveIn` e `WaveNative`.

9.5 Classe `UDPAudioSlave`

La classe, si occupa di costruire e attivare il thread `UDPSlaveThread`, che esegue le seguenti operazioni:

1. costruire un socket per i pacchetti audio e coordinate mouse, registrarsi un indirizzo multicast ed a una porta, entrambe passate come parametri dall'utente
2. ricevere i pacchetti audio e coordinate mouse
3. traformare le coordinate in numeri interi
4. verificare il numero di sequenza
5. inserire i pacchetti audio nella coda condivisa con `audioComponent`
6. inserire i pacchetti delle coordinate del mouse nella coda condivisa con `VisForm`

10. Considerazioni

10.1 Calcolo della banda

Il calcolo della banda utilizzata, avviene, assumendo che i keyframe siano ogni 0 secondi, ovvero che sia sempre trasmessa tutta l'immagine. Questa scelta è dettata dal fatto che a priori non si può valutare il comportamento dell'utente che agisce sul contenuto dello schermo. Queste azioni non sono deterministiche. Il calcolo, tiene però conto di:

- dimensioni delle immagini
- qualità delle immagini
- banda audio
- numero di frame al secondo

10.2 Interazione tra codice gestito e non gestito

Nel progetto, è stato necessario fare ricorso a due DLL dell'ambiente Win32: una per la cattura delle immagini del video `gdi32.dll`, e la seconda per la gestione dell'audio `winmm.dll`. Queste, sono elementi di codice non gestito, non implementate nella piattaforma .NET, si è reso necessario importarle nell'ambiente a codice gestito.

Per eseguire questa procedura è stato utilizzato il metodo Platform Invoke utilizzando le funzionalità offerte da C# in `System.Runtime.InteropServices`. In particolare viene definita una funzione che è un "wrapper" a quella non gestita. Inoltre, durante l'importazione delle strutture è necessario porre attenzione all'ordine sequenziale dei campi. Le applicazioni di queste considerazioni al progetto sono illustrate di seguito:

Libreria `gdi32.dll`*, alcuni esempi*

```
[System.Runtime.InteropServices.DllImportAttribute("gdi32.dll")]

            private static extern bool BitBlt(…);

[System.Runtime.InteropServices.DllImportAttribute("gdi32.dll")]

            private static extern IntPtr  CreateDC(…);
```

Libreria `winmm.dll`*, alcuni esempi*

```
[DllImport(mmdll)]

            public static extern int waveOutGetNumDevs();

[StructLayout(LayoutKind.Sequential)]
      public class WaveFormat {…}
[StructLayout(LayoutKind.Sequential)] public struct WaveHdr{…}
```

10.3 Prestazioni e risorse impiegate

Le prestazioni sono valutate nei ritardi di visualizzazione delle immagini e della riproduzione dei buffer audio. All'incirca, utilizzando le impostazioni di default, i ritardi si possono valutare in circa 2 secondi. Agendo sulle dimensioni delle immagini, sulla qualità e sul keyframe è possibile diminuire questo valore. L'audio dipende dalla modalità di campionamento e dai buffer in registrazione e ascolto utilizzati. Le caratteristiche di campionamento sono 11025 Hz, 16 bit ad un solo canale, mentre i buffer impiegati sono 1 per la registrazione ed 1 per l'ascolto. Aumentare il numero di buffer, permette di introdurre un certo ritardo, ma consente di avere un discorso più fluido.

In termini di memoria utilizzata e percentuale di CPU, si può osservare che con un microprocessore di ultima generazione (Intel Pentium 4 640) si attestano intorno a 30-40 MB (memoria) e 10-12 % (cpu). L'utilizzo della memoria varia (da 30 a 50 MB) con microprocessori meno recenti (Intel Celeron 1 GHz), l'utilizzo della CPU è piuttosto elevato. Il carico della CPU, da osservazioni effettuate, si può attribuire quasi esclusivamente alle catture video. Le misurazioni sono state effettuate utilizzando sullo stesso calcolatore una istanza di trasmettitore ed una di ricevitore. Impiegando calcolatori diversi (per trasmettitore e ricevitore) si osserva che il trasmettitore è l'entità che richiede risorse maggiori.