UNIVERSITA' DEGLI STUDI DI BOLOGNA

FACOLTA' DI INGEGNERIA

Corso di Laurea in Ingegneria Informatica

Reti di Calcolatori

Handoff di flussi multimediali basato su proxy.

Architettura basata su proxy per lo spostamento di dati e sessione.

Candidato:

MATTEO TOMMASO PETRI

Relatore:

Chiar.mo Prof. Ing. ANTONIO CORRADI

Correlatore:

Dott. Ing. LUCA FOSCHINI

Anno accademico 2004-2005

INDICE

PAROLE CHIAVE

———

Wireless Internet

Streaming video

Proxy

Bufferizzazione

Agenti Mobili

HandOff

Introduzione

L’attuale società è caratterizzata dal crescente bisogno di informazione e dalla relativa risposta tecnologica, disponibile a più livelli grazie soprattutto al calo dei prezzi dei dispositivi di connessione individuale, di cui sono un esempio cellulari di nuova generazione e palmari, e l’introduzione di reti wireless con costi di copertura inferiori rispetto alla rete via cavo.

I servizi offerti sono eterogenei come l’utenza che li richiede, si va dal lavoro all’intrattenimento personale e domestico. Esempi di questi servizi sono il Video on Demand (VoD), la videoconferenza, la videosorveglianza e la televisione interattiva.

Per offrire un servizio costruito sulle esigenze dell’utente la risposta tecnologica si è arricchita del concetto di mobilità del servizio. Ad esempio con la tecnologia mobile si vuole consentire a numerosi utenti in movimento, connessi a diversi punti della rete con terminali di accesso eterogenei, di ricevere un servizio multimediale.

Questo scenario rende necessario uno strato Applicativo capace di gestire lo streaming video e i nodi che questo coinvolge.

Lo streaming video pone molti problemi: per prima cosa è necessario che non si verifichino salti nella visualizzazione del flusso di frames (PlayBack Starvation), inoltre, nel caso di un utente mobile, si deve tenere conto della eventuale disconnessione casuale oppure del cambio di “cella” con la conseguente disconnessione dalla precedente e riconnessione alla nuova (handoff) e soprattutto

dell’alta probabilità di errori tipica di una connessione wireless che, data la restrizione sui tempi imposta dallo streaming, lascia poco spazio per una eventuale ritrasmissione.

Per mantenere fluido lo streaming e gestire le momentanee disconnessioni dal servizio sono stati progettati componenti in grado di spostarsi sulle zone di copertura raggiunte dall’utente, seguendo i suoi spostamenti. Questi componenti ricevono il servizio richiesto dall’utente, a cui lo forniscono a loro volta attuando meccanismi finalizzati a realizzare una sorta di vicinanza del servizio stesso.

Si descriveranno nel primo capitolo alcune problematiche che affliggono i sistemi mobili, in particolare verranno approfonditi i concetti di handoff e gestione della sessione. Nel secondo sarà introdotta la tecnologia utilizzata per realizzare questo lavoro, mentre una analisi specifica del problema verrà fornita nel terzo capitolo. Il quarto capitolo descrive le caratteristiche progettuali dei componenti realizzati e la modifica di quelli già esistenti. Infine, nell’ultimo capitolo, saranno illustrati l’implementazione del progetto e i risultati ottenuti.

1. Problematiche dei sistemi mobili

1.1 Continuità del servizio in applicazioni multimediali

Quando si parla di multimediale in genere si intende la combinazione di due o più media continui, cioè media che devono essere riprodotti in sincrono e con continuità durante un intervallo di tempo definito, di solito con qualche interazione d'utente. I due media sono generalmente audio e video.

Per trattare contenuti multimediali è necessario quindi disporre di un tipo di servizio continuo, cioè un servizio che trasmette un flusso di dati con una frequenza costante e adotta strategie volte a evitare interruzioni casuali del flusso.

La trasmissione di contenuti multimediali avviene nell'ambito della sessione tra il fornitore del servizio e l'utente. Per sessione si intende tutto il traffico di dati che le due entità si scambiano in un intervallo di comunicazione, intervallo definito tra il momento in cui viene effettuata la prima richiesta e il termine delle richieste. La sessione si svolge come una conversazione: si ha una prima fase di riconoscimento tra le parti e prosegue con una fase di scambio di informazioni.

Per streaming multimediale si intende l'insieme delle operazioni necessarie al reperimento, al recapito e alla presentazione di un flusso dati multimediale. Lo streaming video è la modalità operativa di una applicazione che permette la visualizzazione delle immagini mentre il trasferimento dei dati é ancora in corso, diversamente quindi dalla modalità download and play che richiede l'attesa della fine

del trasferimento del file. Grazie all'utilizzo di buffer in trasmissione e ricezione é possibile ottenere una vicinanza, una trasmissione real-time che permette anche di usufruire di eventi live, come può essere ad esempio un evento televisivo.

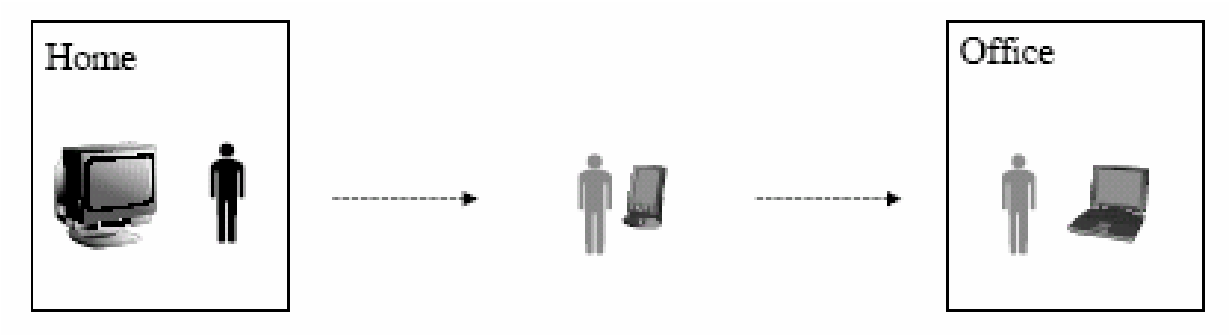
I servizi che operano in modalità streaming possono essere del tipo richiesta per una trasmissione di un video memorizzato su supporto (VoD) ad esempio (parlando di tv, un telegiornale precedente), richiesta per una trasmissione di filmati live, o una conversazione telefonica tramite connessione Internet (VoIP), servizio che richiede vincoli di interattività ancora più stringenti rispetto al VoD.

Questo lavoro di tesi si concentrerà su un servizio di tipo VoD: l'utente richiede solo saltuariamente poche operazioni a chi fornisce il servizio, come l'inizio, l'interruzione o il riavvolgimento del video.

Il problema della trasmissione in real-time su rete IP, come Internet, é che questo tipo di rete non é stato progettato per il trasferimento e la visualizzazione di dati in modalità streaming, in quanto é basata su un servizio best effort che non effettua azioni per garantire l'affidabilità della trasmissione. Per questo motivo la qualità del servizio (QoS) deve essere gestita sopra al livello di rete. La QoS è caratterizzata da un insieme di parametri specifici che consentono all'utente del servizio di specificare sia i valori richiesti che quelli accettabili e minimi per i vari parametri di servizio.

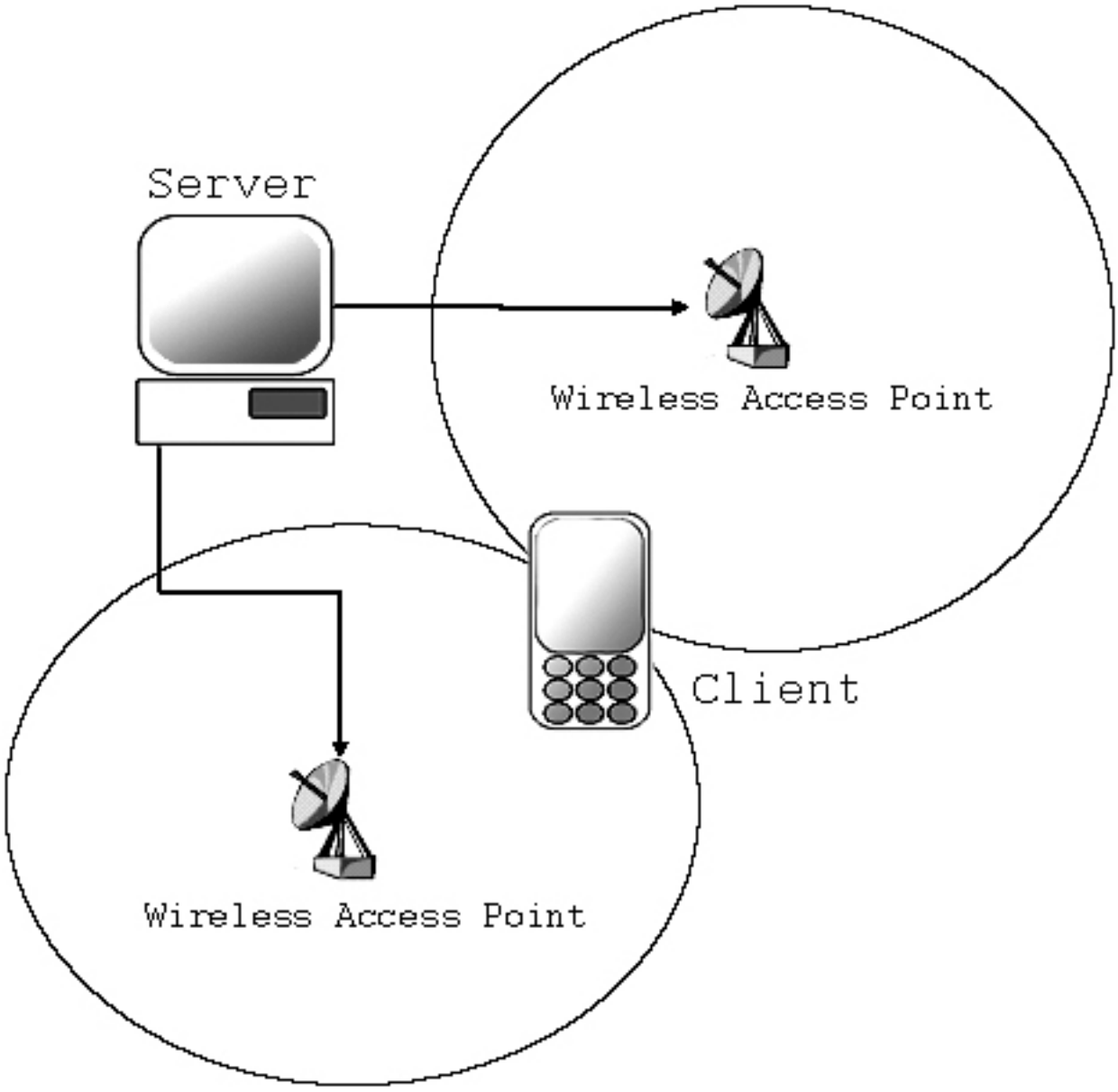
Un utente può effettuare due tipologie diverse di movimento: nomadico o roaming.

Nel movimento nomadico lo spostamento avviene tra più terminali fissi ed è l'utente che fisicamente si sposta da un dispositivo all'altro, si parla perciò di mobilità di utente. La sessione di lavoro viene trasferita da un terminale all'altro in modo che, quando l'utente arriva, può accedere dal nuovo terminale alla fruizione del contenuto multimediale dal punto in cui era stata interrotta.



Esempio di utente mobile, spostamento nomadico

Il movimento di tipo roaming invece si presenta nel caso in cui l'utente si muova assieme al suo terminale in una rete wireless ed è il tipo di movimento preso in considerazione nel nostro scenario. Durante il roaming il dispositivo passa da una zona di copertura del segnale (cella) all'altra connettendosi di volta in volta al Wireless Access Point (WAP) relativo, si parla in questo caso di mobilità di terminale perchè è il terminale che si sposta tra i punti di accesso.



Esempio di terminale mobile, roaming

1.2 Problemi della sessione

I principali problemi nella realizzazione di servizi multimediali, e quindi continui, in un sistema mobile sono inerenti alla gestione della sessione. Nel nostro caso consideriamo un dispositivo wireless che è connesso con un Wireless Access

Point, quando lascia la zona coperta da quello a cui era connesso per passare alla zona coperta da un altro access point si ha una disconnessione dal precedente ed una connessione al nuovo.

Il primo problema è quello di spostare una sessione aperta da una località all'altra e viene detto handoff di sessione. Durante l'handoff la momentanea disconnessione viene percepita come una interruzione del flusso video.

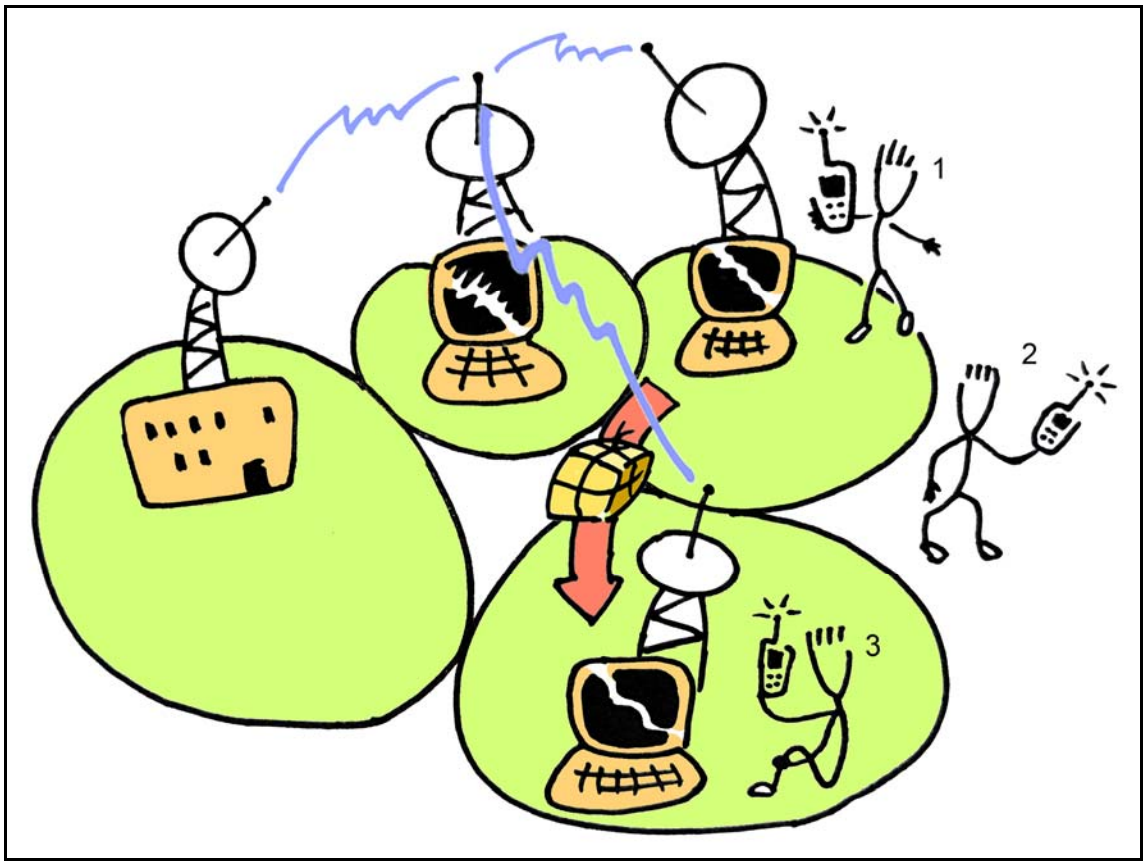
Dopo lo spostamento migratorio anche le risorse utilizzate devono poter essere spostate, risorse che possono avere stato e perciò, una volta migrate, devono tornare a funzionare dall'ultima posizione raggiunta.

L'altro problema è legato alla necessità di gestire queste migrazioni in modo da conservare la sessione, mantenendo i dati relativi a un utente anche in seguito a un suo spostamento, e ristabilire la connessione tra le entità coinvolte. Infatti, nel caso in cui l'utente cliente sia connesso ad una entità servitore e decida di spostarsi, quest'ultima deve aggiornarsi rispetto alla nuova posizione del cliente per continuare a fornire il servizio. Ci si rende conto perciò di come sia necessario disporre di una struttura dove memorizzare una molteplicità di informazioni utili a identificare e rintracciare un utente e i servizi disponibili che va oltre al sistema DNS, dove al nome di un servizio corrisponde semplicemente il suo indirizzo.

Considerando che la trasmissione tra un server e un client in un sistema wireless può passare attraverso molteplici access point nasce l'esigenza di avere un'entità attiva in corrispondenza di ogni access point che si trova sul percorso che

collega il server al client. Queste entità, dette proxyes, formano una catena che porta il servizio ‘vicino’ al client, offrendo una migliore continuità dello streaming.

La catena di entità coinvolte nel servizio rappresenta un modello a delega: il server che fa streaming si occupa della trasmissione alla zona più vicina dove un proxy continua lo streaming verso il componente a valle, che può essere un altro proxy o il client finale.



Catena di proxy e installazione del nuovo proxy in seguito allo spostamento migratorio.

La figura in alto mostra in verde le zone coperte da un access point. Il server è rappresentato dall’edificio giallo mentre i proxyes dai computer. Quando l’utente, che

riceve lo streaming su un dispositivo mobile, cambia zona (spostamento 1-2-3 di figura), anche il percorso per raggiungerlo cambia, per cui è necessario spostare i proxyes e i dati relativi alla sessione sugli access point che formano il nuovo percorso.

1.3 Obiettivo della tesi

Questo lavoro di tesi si inserisce nel progetto di realizzazione di un servizio di VoD su rete wireless. Come descritto in precedenza offrire un servizio multimediale fluido significa garantire la continuità del servizio anche quando la mobilità di terminale provoca rapidi aggiornamenti nel sistema, cioè durante la fase di handoff.

Il nostro lavoro si pone l'obiettivo di gestire il mantenimento della sessione attraverso una catena di proxy che uniscono il server con il client e in caso di handoff spostano la sessione sul nuovo Access Point.

Questo significa spostare tutte le informazioni da un Access Point all'altro, informazioni che comprendono i dati relativi al riconoscimento dell'utente, alle sue preferenze e allo stato del flusso trasmesso.

Inoltre si vuole evitare di perdere tutta la parte di video già inviato sul proxy e non ancora trasmesso al client, risparmiando richieste di ritrasmissione al server che dispone del contenuto multimediale. L'utilizzo di un sistema di proxy permette infatti di caricare parti del video in strutture di bufferizzazione che poi verranno inviate al cliente. Durante l'handoff i frames non ricevuti dal cliente, ma già inviati dal server e

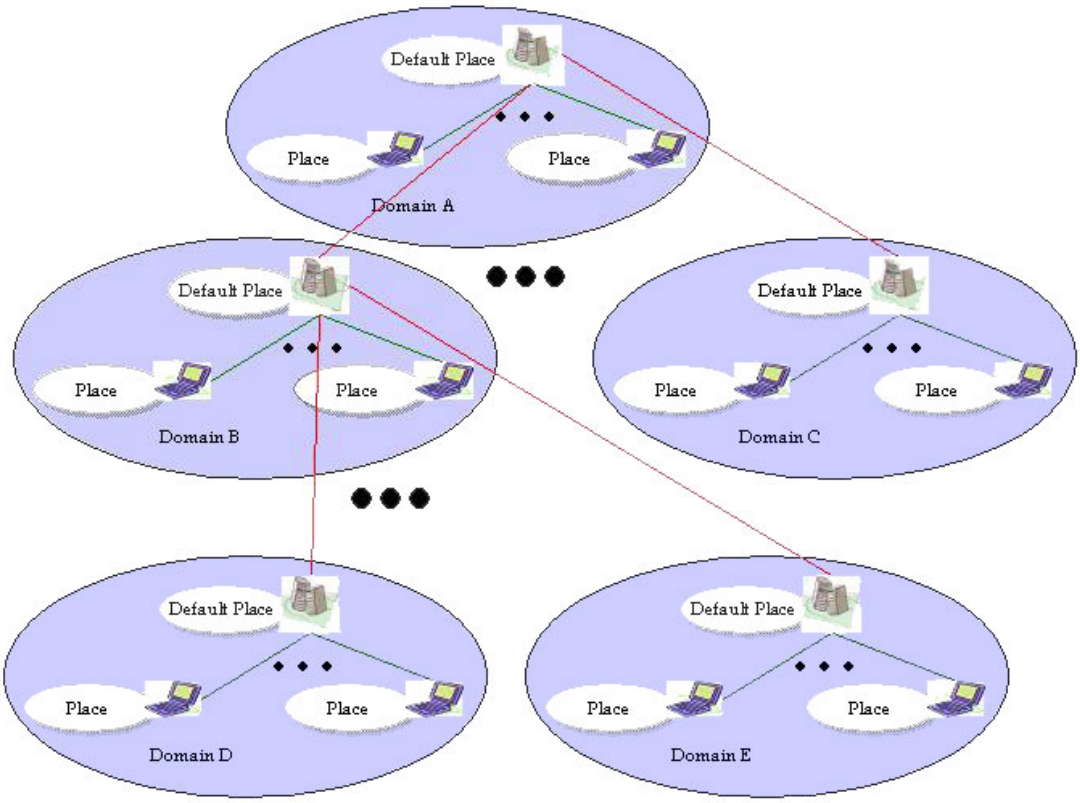
che sono stati memorizzati su un proxy, saranno spostati sul nuovo Access Point senza essere persi e da qui il cliente continuerà a riceverli con continuità.

Infine è necessario stabilire sul proxy un protocollo di funzionamento che permetta di ristabilire la connessione tra client e server in modo da riprendere la trasmissione immediatamente dopo lo spostamento migratorio dello stesso proxy e del client.

2 Tecnologia e infrastrutture utilizzate

2.1 SOMA

SOMA (Secure and Open Mobile Agent) è un framework ad agenti mobili basato su Java. E' stato progettato per fornire meccanismi di sicurezza e di interoperabilità con componenti eterogenei, e per offrire una visione scalabile dello scenario globale. Realizza infatti un'astrazione di rete definendo dei luoghi (places) che sono uniti tra loro in modo gerarchico, ad albero, con nodi e foglie.



Esempio di rete SOMA

Le foglie sono le estremità dell'albero dove possono essere mappate entità client o server, la distanza tra il servizio e l'utilizzatore è calcolata in base al numero di nodi che li separano.

I places rappresentano gli access point della rete. Sulla struttura di SOMA è possibile ricercare informazioni su contenuti multimediali, inviare ricevere e utilizzare dati e definire i profili degli utenti. Queste informazioni vengono memorizzate in piani (plans). I piani sono usati dagli agenti mobili per spostarsi all'interno di questa rete e riconoscere all'interno anche le diverse tipologie di entità: proxy, client e server.

2.2 Indirizzamento di un agente

Gli agenti messi a disposizione da SOMA si muovono sull'astrazione di rete che questo framework fornisce e implementano i metodi dell'interfaccia *SOMA.agent.Agent*.

I metodi usati per configurare un'agente in ordine di utilizzo sono *putArgument()* e *go()*. Il primo carica l'agente con gli argomenti, che possono essere oggetti, da trasportare sul nodo destinazione, mentre il metodo go() lancia l'agente. I parametri di questo metodo sono il nodo target, che può appartenere ad un dominio qualsiasi e il nome del metodo con cui verra' riattivato l'agente.

2.3 MUM

Mantenere una sessione aperta durante la fase di handoff, fase in cui il cliente si muove attraverso gli access point wireless, significa offrire un servizio continuo, indispensabile per servizi di tipo multimediale. MUM (Mobile agent based Ubiquitous multimedia Middleware) è una infrastruttura middleware progettata a questo scopo e offre supporto a servizi VoD su rete a filo o wireless per garantire QoS e continuità del servizio.

L'utente può richiedere un filmato e riceverlo sul supporto multimediale a sua disposizione, che può essere un computer, un palmare, un cellulare o altro e muoversi durante la ricezione, senza essere vincolato a un unico luogo di fruizione del servizio.

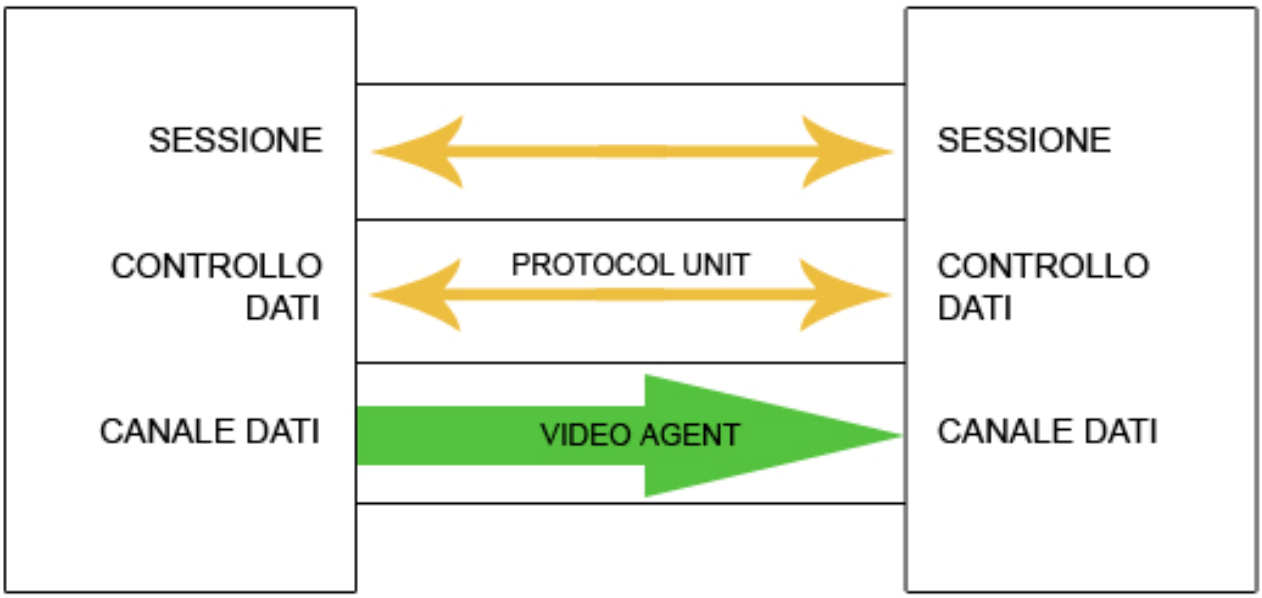
Per mantenere attiva la sessione durante l'handoff MUM utilizza un'infrastruttura di proxies che gestiscono le risorse "inseguendo" il cliente nel suo spostamento e adattando il servizio al suo profilo. I proxies, come illustrato in precedenza, possono essere installati dinamicamente sul percorso tra server e client a seguito di spostamenti di quest'ultimo. La struttura *service path* rappresenta questo percorso e raccoglie tutti i componenti attivi che partecipano alla sessione. Il concetto di path arricchisce l'astrazione di rete fornita da SOMA, su cui MUM è basato.

Il path descrive il percorso dal nodo radice, il più alto nell'albero che rappresenta il punto di partenza della distribuzione del servizio VoD, fino ai nodi interni e alle foglie. I terminali client, con limitate e differenziate risorse locali, sono localizzati sulle foglie di questa struttura. A seguito dello spostamento di un client su

un nuovo nodo foglia il path viene modificato in modo da collegare quest'ultima posizione al nodo radice e i componenti necessari (proxy) vengono installati sui nodi interni descritti nel path.

MUM sfrutta la tecnologia ad agenti mobili per spostare parti di codice dove occorre. Scaricare soltanto il software necessario è indispensabile per più ragioni. In primo luogo dispositivi come cellulari e palmari hanno ridotta capacità di memoria che rende impossibile l'installazione di programmi pesanti e, anche nel caso occorra installare del codice sui nodi della rete, è indispensabile ottimizzare l'utilizzo della memoria. Per esempio, nella fase di handoff, il proxy che si sposta viene eliminato dall'attuale località e installato sulla nuova in modo da consumare la minor quantità di risorse; poi il codice eseguibile scaricato viene riattivato dall'agente stesso dopo la migrazione e mantiene lo stato precedentemente raggiunto.

MUM è stato realizzato in Java, linguaggio di programmazione che offre la possibilità di aggiungere codice dinamicamente. L'utilizzo di agenti mobili è però riservato solo alle operazioni di alto livello che prevedono spostamento di codice, cioè l'installazione dei componenti e la loro configurazione, mentre per le operazioni di localizzazione del servizio, management del profilo e delle risorse si usa il semplice modello cliente-servitore.



Rappresentazione delle connessioni usate per ogni singolo flusso tra due entità comunicanti

MUM usa tre canali per la gestione del flusso multimediale. Lo streaming multimediale avviene attraverso un canale dedicato unidirezionale che porta i dati dal server al client. A questo è associato un canale di controllo gestito automaticamente dal protocollo di trasporto e trasparente al programmatore. Maggiori dettagli di questo protocollo (RTP e RTCP) saranno dati in seguito. Per consentire l'interazione tra utente e servizio MUM implementa un terzo controllo dei dati. Su questo canale viaggiano ad esempio i comandi di avvio e interruzione dello streaming effettuati dall'utente.

2.4 *Architettura di MUM*

MUM è diviso in due livelli logici, *Middleware Mechanism Layer*, che realizza i servizi di base, e *Middleware Facilities Layer*, che comprende strategie e realizza servizi più complessi.

Architettura a livelli di MUM

Il livello Mechanisms realizza i servizi necessari al livello sovrastante come la localizzazione del servizio, ottenuta attraverso l'astrazione di località fornita da SOMA, la gestione del Software, con la memorizzazione di codice e componenti attivi su ogni nodo, la gestione delle risorse, che controlla le risorse in ogni nodo, e la gestine dei profili e delle presentazioni dei contenuti multimediali.

Questo livello utilizza esclusivamente soluzioni basate sul modello cliente/servitore perchè offre servizi che non richiedono una dinamicità tale da rendere necessario l'uso di un potente paradigma come quello ad agenti mobili, come accade invece per il livello Facilities.

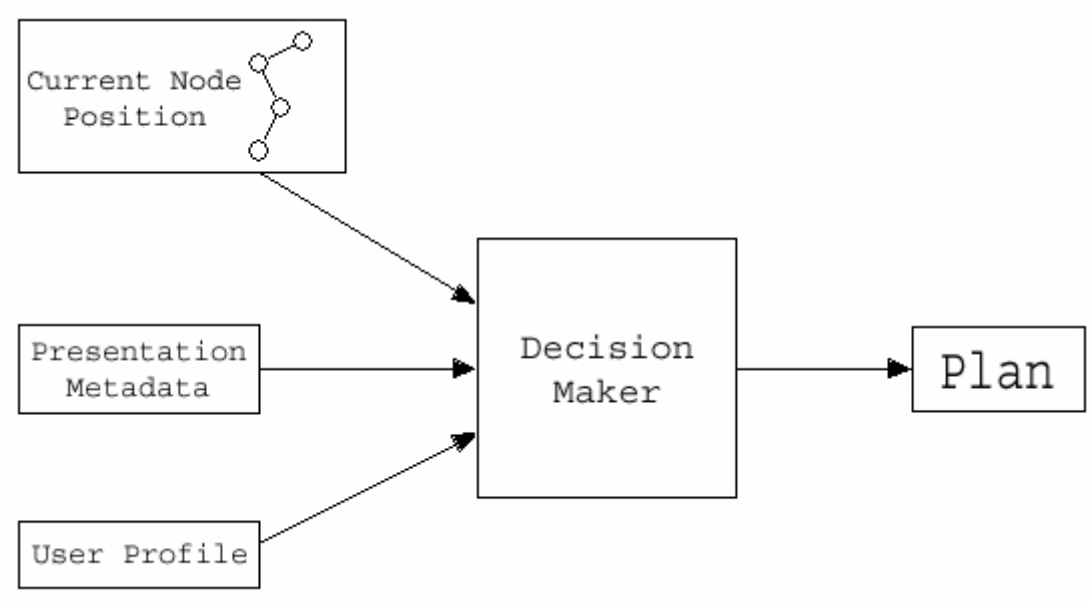
Il livello Facilities rappresenta la parte dinamica di MUM, qui sono infatti utilizzati agenti mobili per realizzare varie strategie in modo autonomo e dinamico.

I servizi offerti da questo livello sono il download del codice, incapsulato nell'agente assieme ai metodi per il reperimento, e il download dello stesso, la configurazione dei service path, che comprende la negoziazione della QoS, la riconfigurazione del sistema a tempo di esecuzione, la gestione del QoS, che viene negoziata inizialmente e può essere rinegoziata a tempo di esecuzione, e la gestione dello spostamento dell'utente, che può essere nomadico o migratorio (a seconda si tratti di rete fissa o mobile).

2.5 Decision Maker

Un *Decision Maker* è usato per incapsulare le strategie per decidere come configurare il *Service Path*, come individuare cioè il nodo attuale e il percorso ottimale tra i nodi della rete fino all'obiettivo dello spostamento.

Utilizzando le funzionalità offerte dal Mechanism Layer ottiene i dati relativi al profilo dell'utente e alla presentazione a cui è interessato ed elabora queste informazioni in un Plan che contiene una o più soluzioni corrispondenti ai diversi formati in cui è disponibile la presentazione, cioè con QoS diversi.

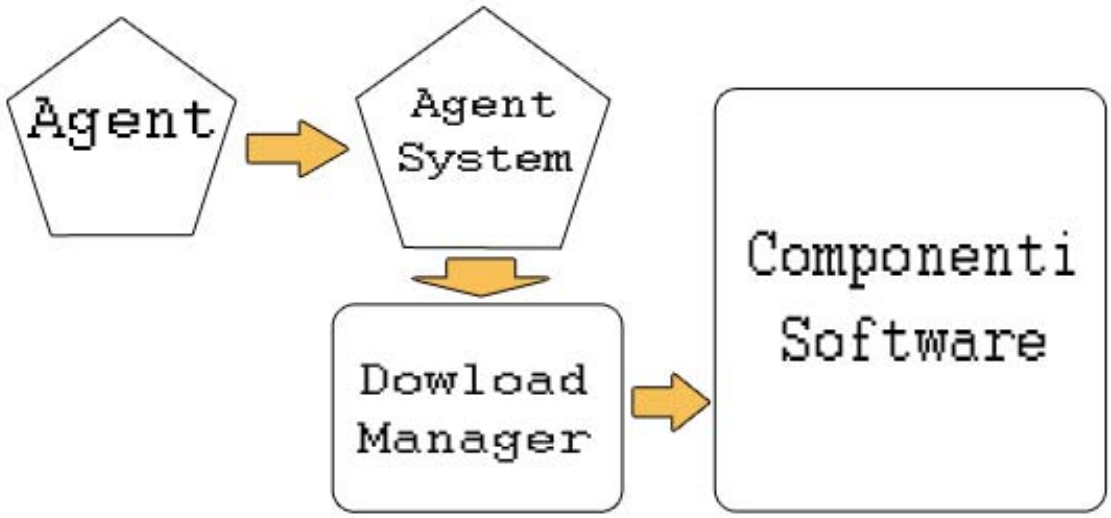


Costruzione di un Plan

2.6 Download del codice nell'agente

Un agente accede a qualsiasi risorsa scaricabile attraverso il *Download Manager*. In questo modo è possibile rendere visibili all'agente solo le risorse desiderate, non è possibile infatti accedere ad esse senza farne richiesta al Download Manager.

Il riferimento a questo componente si ottiene attraverso l'agente di sistema o *Agent System*, unica interfaccia tra il sistema e l'agente, col metodo *agentSystem.getDownloadManager*.



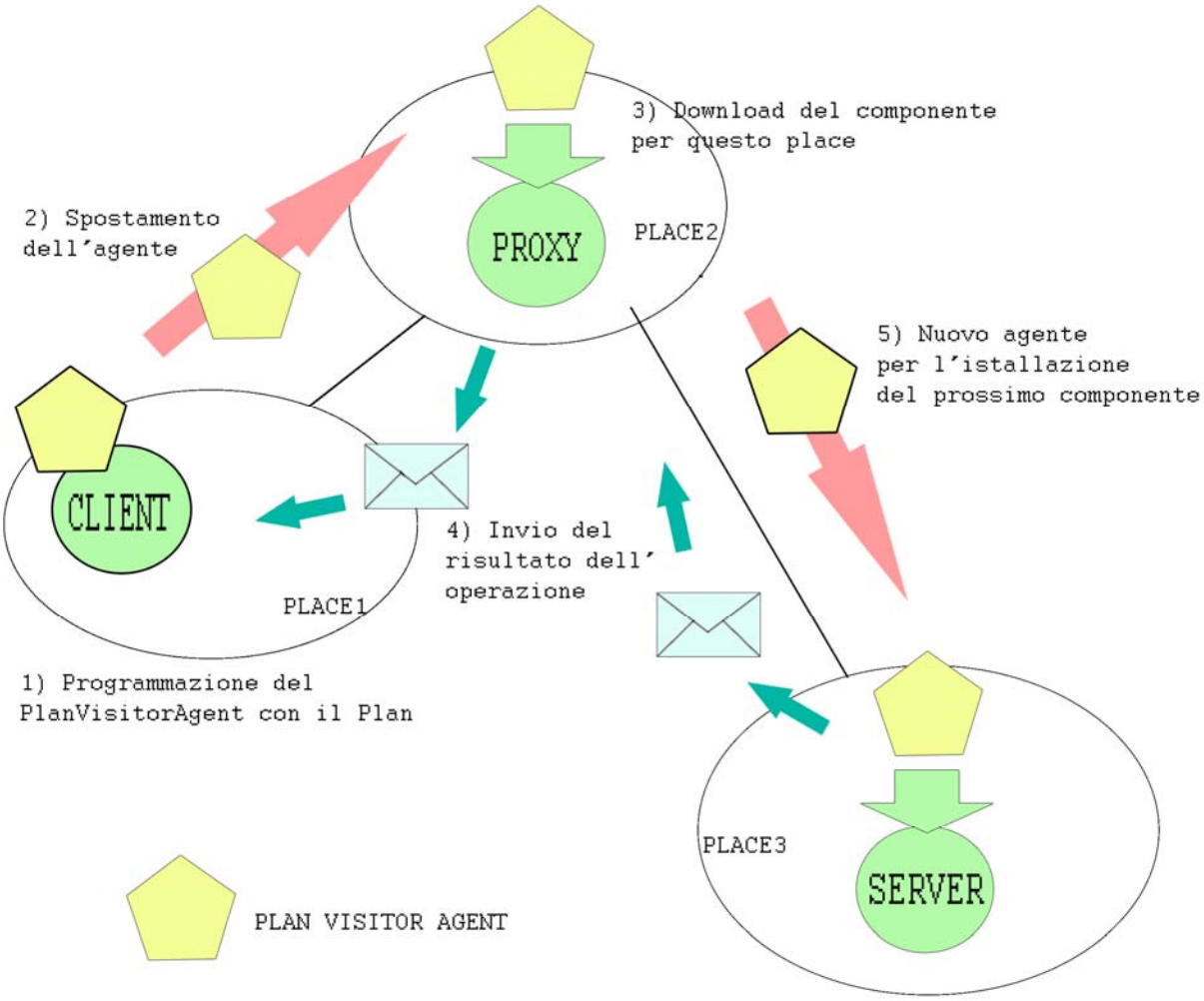
Accesso alle risorse da parte di un agente tramite DownloadManager.

2.6 Plan Visitor Agent

L'agente delegato alle operazioni di installazione delle entità nel sistema è il Plan Visitor Agent.

Questo agente analizza il piano passatogli come parametro e distingue i casi tra piano singolo, multiplo e nomadico (relativo allo spostamento del cliente) secondo il pattern Visitor.

L'agente percorre il path (o i path nel caso si tratti di un piano multiplo) contenuti nel piano e installa su ognuno di essi il codice necessario all'entità da attivare. In particolare, una volta raggiunto il nodo destinazione, l'installazione avviene in più fasi.



Esempio di esecuzione di un plan per opera del PlanVisitorAgent

L'agente crea per prima cosa il piano individuando i componenti scaricabili per questa entità, accessibili attraverso il Download Manager. Passa quindi all'inizializzazione dei componenti scaricati invocandone i metodi init.

A questo punto l'agente lancia un altro Plan Visitor Agent fornendogli tra i parametri il path restante, che contiene solo il percorso non ancora esplorato, e attende che il nuovo agente abbia completato il suo lavoro.

Una volta ricevuto il messaggio dall'agente successivo che notifica l'avvenuta installazione dell'entità seguente, l'agente padre invia a sua volta una notifica positiva all'agente che l'ha creato. Il path viene percorso ricorsivamente dagli agenti e le entità installate: in questo modo si realizza il bind delle entità attive.

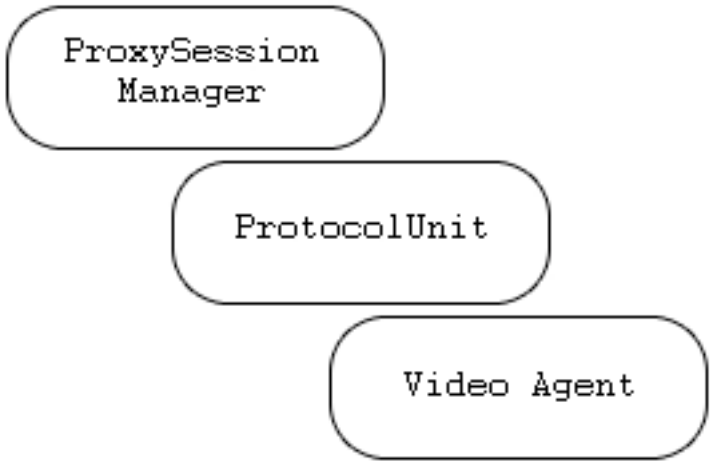
Altri agenti vengono lanciati dal Plan Visitor Agent per completare l'istallazione delle entità (ad esempio il Proxy Agent).

2.7 Componente Proxy

MUM realizza tre tipi di entità attive: proxy, client e server. Dovendo intervenire sul componente proxy è necessario chiarirne il funzionamento.

Il proxy viene installato su un place della rete dal Plan Visitor Agent come le altre entità. Dopo avere scaricato il codice del componente l'agente lo inizializza con due operazioni su livelli architetturali distinti.

A livello Mechanism viene inizializzato il protocollo per la gestione della sessione (ProxySessionManager). Questo a sua volta inizializza due unità di controllo e una dati. Le prime (ProtocolUnit) servono a gestire rispettivamente la connessione con il server e con il client, stabiliscono la connessione RTP da cui ricevono lo stream dal server e ricevono dal canale dati i comandi dell'utente (come play e stop) con cui viene gestito lo streaming. L'unità dati (Video Agent) gestisce il canale dati su cui avviene lo streaming verso il client.



Gestione della connessione strutturata a livelli

A livello Facility il Plan Visitor Agent inizializza un altro agente (ProxyAgent) che rimane attivo per tutta la durata di funzionamento del componente proxy per gestire la QoS.

2.8 Java Media Framework

La tecnologia usata per sviluppare l'oggetto di questo lavoro di tesi è il Java Media Framework, una libreria java per la gestione di oggetti multimediali.

JMF mette a disposizione dello sviluppatore una vasta collezione di API (interfacce di programmazione) per aggiungere contenuti multimediali come audio e video ad applicazioni e applet JAVA ed è specificatamente progettato per sfruttare al meglio le caratteristiche di questo linguaggio.

Il motivo per cui è stato scelto questo framework ed anche il linguaggio java è la sua portabilità su piattaforme diverse che li rendono la soluzione perfetta per un

sistema che deve essere eterogeneo e gestire diversi tipi di terminali, dai computer dotati di una connessione fissa ai PDA dotati di una connessione wireless.

Esistono quattro implementazioni del JMF, una che usa codice scritto puramente in java (Cross Platform Edition, non sempre la più efficiente in quanto la decodifica di molti stream video è un'operazione pesante in termini di cicli di CPU) ed altre (Solaris, Linux e Windows performance pack) che utilizzano plug-in scritti in C e implementati specificatamente per ogni piattaforma.

JMF mette a disposizione del programmatore molti metodi per gestire la distribuzione di un flusso multimediale (ad esempio un video).

In particolare JMF permette di programmare applicazioni multimediali utilizzando componenti ed API ad alto livello già fornite dal FrameWork stesso (ad esempio il Player, classe auto-configurante in grado di effettuare la visualizzazione di un flusso richiedendo solo poche indicazioni da parte del programmatore) che semplificano la programmazione ma limitano l'espressività oppure utilizzare API di più basso livello che hanno una grande potenza espressiva ma la cui programmazione è molto più complessa.

2.9 DataSource

La classe DataSource fornisce un modello semplificato ed astratto del Media che può essere utilizzato come parte della catena di processing o di controllo.

La sua costruzione richiede la specifica di un protocollo e della locazione da cui il Media può essere ottenuto.

Le istanze del DataSource gestiscono il trasferimento del Media controllando una o più istanze di SourceStream, che rappresentano gli Stream del Media.

Pur essendo possibile creare direttamente un DataSource tramite i suoi due costruttori, è in pratica inutile dato che il DataSource base non fornisce nessuna funzionalità, sono le sue sottoclassi (create utilizzando gli appropriati metodi della classe Manager) a possedere le funzionalità desiderate.

I metodi chiave di questa classe sono connect(), disconnect(), start() e stop(); i primi due aprono (o chiudono) la connessione al MediaSource specificato alla creazione del DataSource, gli altri due iniziano (o fermano) il trasferimento dei dati.

I DataSource possono essere classificati in due modi: il primo secondo il modo in cui il trasferimento è realizzato (modalità push o pull) e l'altro riguarda le unità con cui le informazioni sono trasferite (raw o buffer).

I PullDataSource hanno il trasferimento inizializzato e controllato dal client (ad esempio i protocolli http e FTP) mentre i PushDataSource hanno il trasferimento controllato dal server.

Per quanto riguarda il metodo con cui le informazioni sono trasferite, il RawDataSource trasferisce le informazioni come semplice flusso di byte, mentre i BufferDataSource incapsulano le informazioni dentro oggetti di tipo Buffer.

Due categorie per ciascuno dei due assi portano alla presenza di quattro sotto-classi di DataSource: PullDataSource, PushDataSource, PullBufferDataSource, PushBufferDataSource.

Due tipi speciali di DataSource possono essere creati attraverso metodi della classe Manager: cloneable e merged.

Il DataSource cloneable può essere clonato per permettere ad esempio di processare contemporaneamente più copie dello stesso video, mentre più DataSources possono essere unite creando un merged DataSource che contiene tutti i loro SourceStream.

Processare un DataSource significa passarlo a un componente che lo consuma, cioè legge sequenzialmente i pacchetti contenuti nel DataSource. Questa operazione provoca un effetto di trascinamento sul canale dei dati, dove la consumazione di un pacchetto provoca la produzione (invio) di un altro pacchetto da parte del server.

Quindi processare un DataSource significa provocarne lo streaming. I componenti definiti da JMF per questo scopo sono il Processor e il Player (che a differenza del primo realizza anche la visualizzazione del filmato), ma un funzionamento analogo si ottiene processando il DataSource attraverso la catena Demultimplexer-Multiplexer, il cui funzionamento sarà illustrato in seguito.

Ogni DataSource può essere utilizzato da un solo processore per volta ma come detto in precedenza può essere clonato.

2.10 Format

La classe Format permette di descrivere il formato del flusso multimediale in senso astratto; oggetti di tipo Format non forniscono informazioni sulla codifica oppure sulle temporizzazioni.

La classe Format è estesa da 3 classi più specifiche: AudioFormat, VideoFormat e ContentDescriptor. Queste classi sono poi estese da classi specifiche per ogni tipo di formato (ad esempio YUVFormat, H263Format).

Ogni flusso ha associato un ContentDescriptor ed ogni Buffer ha un suo formato che può essere ottenuto attraverso il metodo getFormat().

Ad ogni stadio di processo del flusso è associato un suo formato, ad esempio un filmato è inviato con il formato H263_RTP, poi è decodificato e convertito in YUV, infine è convertito in RGB per poter essere presentato all'utente.

2.11 Buffer

La classe Buffer è il contenitore che permette di trasferire dati da uno stadio di processo ad un altro.

Nel JMF il Buffer rappresenta un singolo frame del flusso multimediale, frame che secondo il livello in cui ci troviamo può essere o un frame applicativo o un frame RTP. Un Buffer contiene solitamente o un frame completo oppure una sua parte oltre ad informazioni come il Formato del Frame, timeStamp (riferito all'istante in cui il

frame dovrebbe essere presentato), lunghezza, numero di sequenza ed altre informazioni (tutte accessibili tramite metodi appropriati).

Il limite della classe Buffer e della sua estensione ExtBuffer è quello di non essere serializzabili. Non è possibile infatti trasformare questi oggetti in flussi di bit in modo da poter essere trasmessi.

2.12 Trasporto dati multimediali e RTP

I pacchetti di dati viaggiano sulla rete grazie ai protocolli di trasporto, che sfruttano i servizi offerti dai sottostanti protocolli di rete come l'IP per far giungere i pacchetti ai loro destinatari. Due sono le implementazioni dei protocolli di trasporto:

TCP (Trasmission Control Protocol)

UDP (User Datagram Protocol)

Il protocollo UDP è un protocollo che utilizza una semantica may-be; in questo modo un pacchetto è inviato una sola volta e non ne è garantito l'arrivo, quindi ogni azione per garantire la Qualità del Servizio (arrivo dei pacchetti, richiesta di rinvio nel caso di perdita dei pacchetti) deve essere fatta a livello applicativo.

Il protocollo TCP è un protocollo che utilizza una semantica at-most-once e realizza una Qualità di Servizio maggiore. Infatti se il pacchetto di dati arriva viene considerato al più una volta e in caso di perdita di pacchetti il reinvio è gestito a livello di protocollo assieme alla gestione dell'ordine dei pacchetti.

In caso di congestione della rete però il protocollo TCP rallenta di molto la sua trasmissione, rendendolo inadatto in questo caso ad un'applicazione in tempo reale come lo Streaming di un contenuto multimediale.

La scelta migliore per applicazioni di questo tipo è il protocollo RTP (Real-time Transport Protocol) accoppiato all'UDP e supportato da JMF.

Il protocollo RTP fornisce metodi per creare un sistema di trasporto end-to-end capace di supportare applicazioni in tempo reale. Inoltre consente di interfacciare il flusso video - che ha delle caratteristiche di sincronizzazione molto stringenti, ad esempio con il flusso audio - con la rete Internet. RTP non dà garanzie di QoS ma fornisce solo strumenti per gestire la QoS a livello applicativo e supporta il monitoraggio della trasmissione.

A seconda del tipo di applicazione, poi, gli strati inferiori della pila protocollare possono variare in relazione alle specifiche delle varie classi di Qualità del Servizio.

I pacchetti RTP consistono di un Header di 12 Byte seguito da un Payload di lunghezza dipendente dal media trasportato.

L'Header contiene informazioni come il tipo di Payload, il time stamp, il numero di sequenza e l'identificatore della sorgente.

Al protocollo RTP (che si occupa del trasporto) è affiancato il protocollo RTCP (RTP Control Protocol), protocollo di controllo che permette di monitorare la QoS.

Il protocollo RTCP descrive vari tipi di pacchetti, che contengono nell'Header sempre un Source Descriptor.

Nell’implementazione attuale del JMF l’invio dei pacchetti RTCP è automatizzato ed il programmatore non può gestirlo; quindi per inviare informazioni di controllo specifiche dell’applicazione tra C/S viene usata una terza connessione di controllo oltre a quella su cui è inviato lo stream e quella RTCP.

2.13 Package UNIBO

Il package Unibo è un package java creato dal DEIS che contiene classi che implementano i componenti necessari per creare e gestire le catene di PlugIn e permettere il controllo al livello del singolo frame.

Attraverso i componenti offerti da tale package è possibile implementare meccanismi di Buffering oppure variare dinamicamente la stessa catena di PlugIn.

Una classe di importanza fondamentale è il CircularBuffer, che implementa l’astrazione di un buffer circolare che può essere interposto fra passi successivi della catena.

Gli elementi del CircularBuffer sono singoli Frame (istanze di ExtBuffer, classe che estende Buffer fornendo la capacità di gestire dati in formato nativo). Questa classe gestisce al suo interno le problematiche legate alla gestione della mutua esclusione in caso di accesso concorrente.

La classe è strutturata come un buffer circolare con due puntatori: uno alla prima cella di memoria in cui è possibile scrivere ed uno alla cella da cui è possibile leggere.

Per inserire ed estrarre frames dal buffer si usano altri componenti di questo package. Il primo elemento della catena è il parser. Una volta ricevuto il flusso in ingresso il parser, o demultiplexer, trasforma i dati ricevuti in singoli frames. Per questa operazione è necessario conoscere i formati supportati.

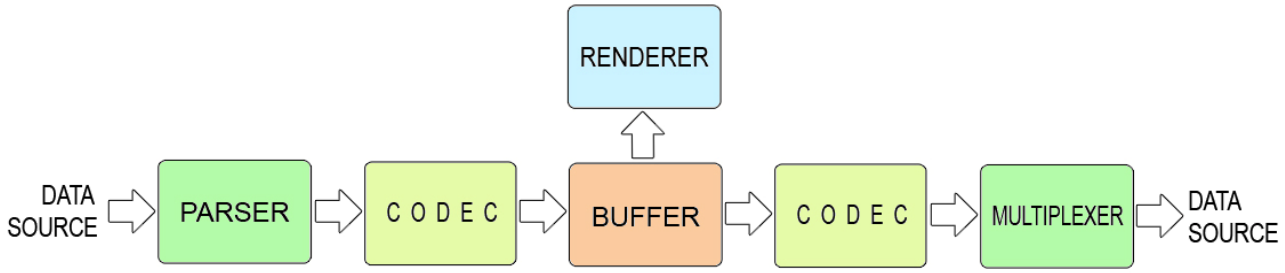
Successivamente uno o più codec trasformano i frames da un formato all'altro. Un Codec è un plugin che processa un frame in ingresso e produce un frame in uscita; nel JMF non è solo una classe che trasforma il formato del buffer di ingresso in quello desiderato sul Buffer di uscita ma ogni operazione che viene eseguita su un Buffer in ingresso e produce un Buffer in uscita cade sotto la categoria di Codec.

A questo punto della catena è possibile operare modifiche sui frames come ridimensionamenti o alterazioni della risoluzione, è possibile anche passarli a un Renderer che procede alla loro visualizzazione.

Sempre a questo livello è possibile agire per memorizzare i frames nel CircularBuffer.

Per la trasformazione inversa i frames vengono processati da altri codec e un Multiplexer ricrea un Data Source che può essere trasmesso sul canale RTP.

In realtà nel nostro lavoro la catena di PlugIn è composta soltanto da un Parser che decompone il DataSource nei singoli pacchetti RTP per permettere il loro inserimento nel buffer circolare e da un Multiplexer che riunisce i pacchetti RTP in un nuovo DataSource.



Catena di PlugIn

Dovendo operare sul componente proxy non è infatti necessario utilizzare codec per la trasformazione del formato dei frames e la successiva visualizzazione, ad eccezione di un componente Renderer creato solo per testare il riempimento del buffer sul proxy, ma ci si concentra esclusivamente sul problema della bufferizzazione dei pacchetti RTP.

3. Analisi del problema specifico

3.1 Buffering sul proxy

Questo lavoro si inserisce nell'infrastruttura di MUM con lo scopo di dotare le entità proxy di un buffer circolare in grado di far fronte a eventi di handoff della sessione anche sul lato del proxy. Il client infatti è già dotato di un sistema di bufferizzazione che gli permette di continuare a visualizzare il filmato anche durante le temporanee perdite di connessione, caratteristiche di una rete wireless, a patto che il periodo di 'buio' non superi la durata del video bufferizzato.

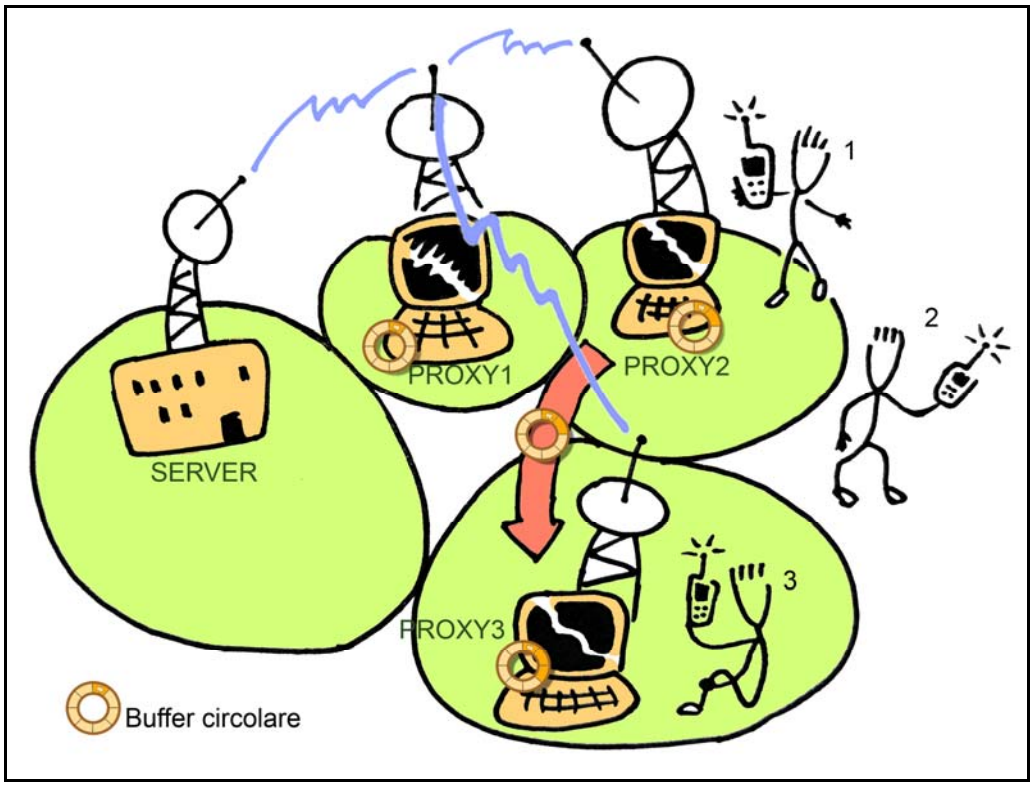
Un buffer si carica ciclicamente dei frames ricevuti e contemporaneamente, con un processo di sincronizzazione, manda quelli letti sullo stream.

La scelta di dotare le entità proxy coinvolte in una sessione di un sistema di bufferizzazione è necessaria al fine di mantenere una buona costanza del flusso video per vari motivi.

In primo luogo, nel caso in cui la comunicazione tra server e proxy peggiori, ad esempio per un sovraccaricamento momentaneo del server, il proxy può continuare a spedire la parte di filmato memorizzata al client che in questo modo non percepisce il rallentamento.

Inoltre nello scenario descritto, relativo allo spostamento di un proxy, e per i motivi visti in precedenza (ottimizzazione delle risorse di rete), è importante che questo buffer non venga perso a fronte di tale migrazione quando non sia stato ancora

completamente consumato. Infatti in caso di handoff della sessione, quando il proxy viene trasferito su un altro host della rete deve chiedere al server da cui riceve il flusso la ritrasmissione dei frames memorizzati dal proxy precedente, ma che non sono ancora giunti al client. La soluzione a questo problema quindi consiste nel trasferire l'intero buffer memorizzato su un proxy nel successivo a fronte dell'handoff.



Spostamento del buffer durante l'handoff

.

Questa operazione richiede l'utilizzo di un agente mobile che oltre ad installare sul nodo un nuovo proxy vi trasferisce contemporaneamente anche il buffer agendo da mezzo di trasporto. Da qui il buffer finirà di trasmettere il flusso interrotto.

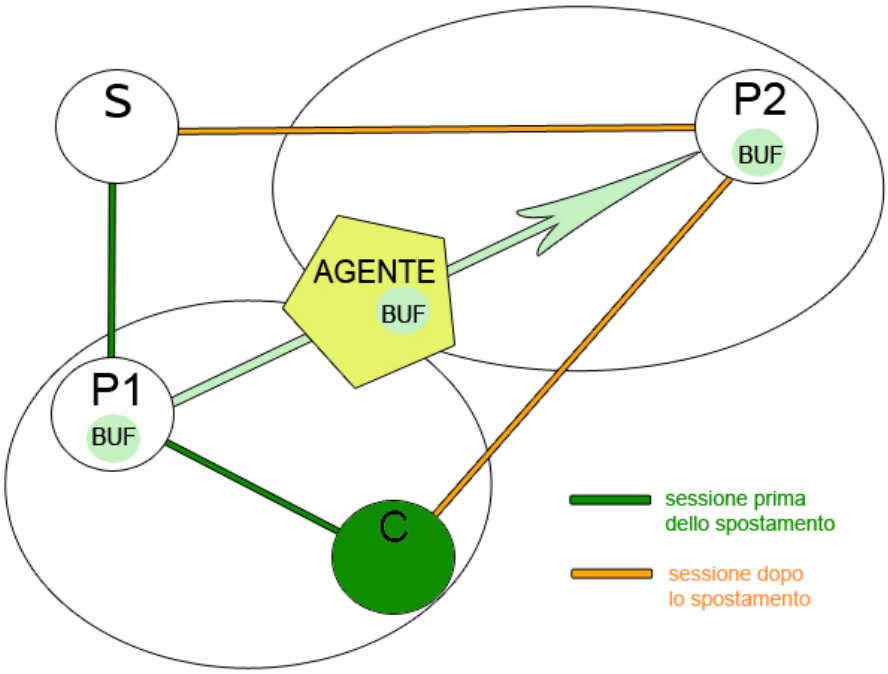
Il buffer, che può essere trasportato sul nuovo proxy e qui riattivato, permette la riattivazione dello streaming verso il client con il filmato che ha in memoria senza attendere la riconnessione al server.

Un altro motivo per cui è importante conservare il buffer in seguito all'evento di handoff è che l'operazione di bufferizzazione coinvolge una catena di PlugIn che ottiene i frames dal canale di trasmissione RTP, operazione che risulta costosa sia in termini di trasmissione che di calcolo.

In una infrastruttura come quella realizzata da MUM, dove la risorsa viene portata vicino al cliente per avere una migliore continuità del flusso, avere un sistema di bufferizzazione mobile permette di ottenere un servizio fluido anche dopo eventi di handoff.

3.2 Scenario del problema

Lo scenario del problema è quello indicato nella figura del capito precedente: il client è connesso al server attraverso un proxy. Nella realtà è il client che con il suo spostamento migratorio provoca un evento di handoff e quindi causa lo spostamento del proxy con conseguente riaggiornamento del service path, ma in questo lavoro interessa isolare e studiare il comportamento del proxy durante lo spostamento. Per questo, trovandoci in un sistema mobile e dovendo gestire lo spostamento di un buffer da un proxy all'altro, ipotizziamo che sia il proxy a spostarsi, come illustrato nella figura sottostante.



Scenario semplificato del problema.

Server e client rimangono nelle rispettive posizioni, perciò il proxy deve gestire la momentanea perdita di sessione che avverrà dal passaggio da un host, o da una cella, all'altra e riattivarla a migrazione avvenuta.

In particolare il nuovo proxy dovrà essere installato passando come argomento l'indirizzo di sessione col server per riprendere immediatamente la comunicazione.

Dovrà inoltre contrattare un nuovo indirizzo per la connessione RTP col client. Questa seconda operazione è svolta dalla *Protocol Unit* di ciascuna entità.

Per lo spostamento di codice e la successiva riattivazione è necessario creare componenti ad hoc sfruttando la tecnologia ad agenti mobili.

3.3 *Buffer Agent*

Per installare un componente proxy si potrebbe utilizzare il Plan Visitor Agent indirizzato con un opportuno plan nomadico. Tuttavia, per non caricare questo agente di ulteriori compiti si è scelto di crearne uno nuovo con funzionalità analoghe all'originale, ma con lo scopo specifico di installare un proxy e e trasportare un oggetto buffer.

3.4 Analisi delle problematiche

Lo spostamento di un oggetto con stato e la successiva riattivazione dal punto in cui è iniziata la migrazione richiede l'utilizzo di strutture *serializzabili*.
Queste strutture, definite nel linguaggio JAVA, sono oggetti con metodi e attributi che possono conservare uno stato interno e hanno la peculiarità di poter essere trasformati in una semplice sequenza di bit attraverso il processo di serializzazione.
Il flusso di bit così ottenuto può poi essere ritrasformato nell'oggetto originale (marshalling) attraverso un'operazione di cast.

Un oggetto serializzabile deve contenere metodi e attributi primitivi come interi, float, ecc o istanze di classi a loro volta serializzabili. Nel nostro caso il buffer deve potere essere serializzato e perciò come visto sopra deve contenere solo oggetti serializzabili.

Un primo problema nasce dal fatto che la struttura messa a disposizione dal supporto JMF per la memorizzazione dei frames non è serializzabile perciò il buffer, che raccoglie una collezione di ExtBuffer, non potrà essere spostato.
E' quindi necessario ricreare questa e altri API di alto livello in modo da renderle funzionali all'utilizzo.

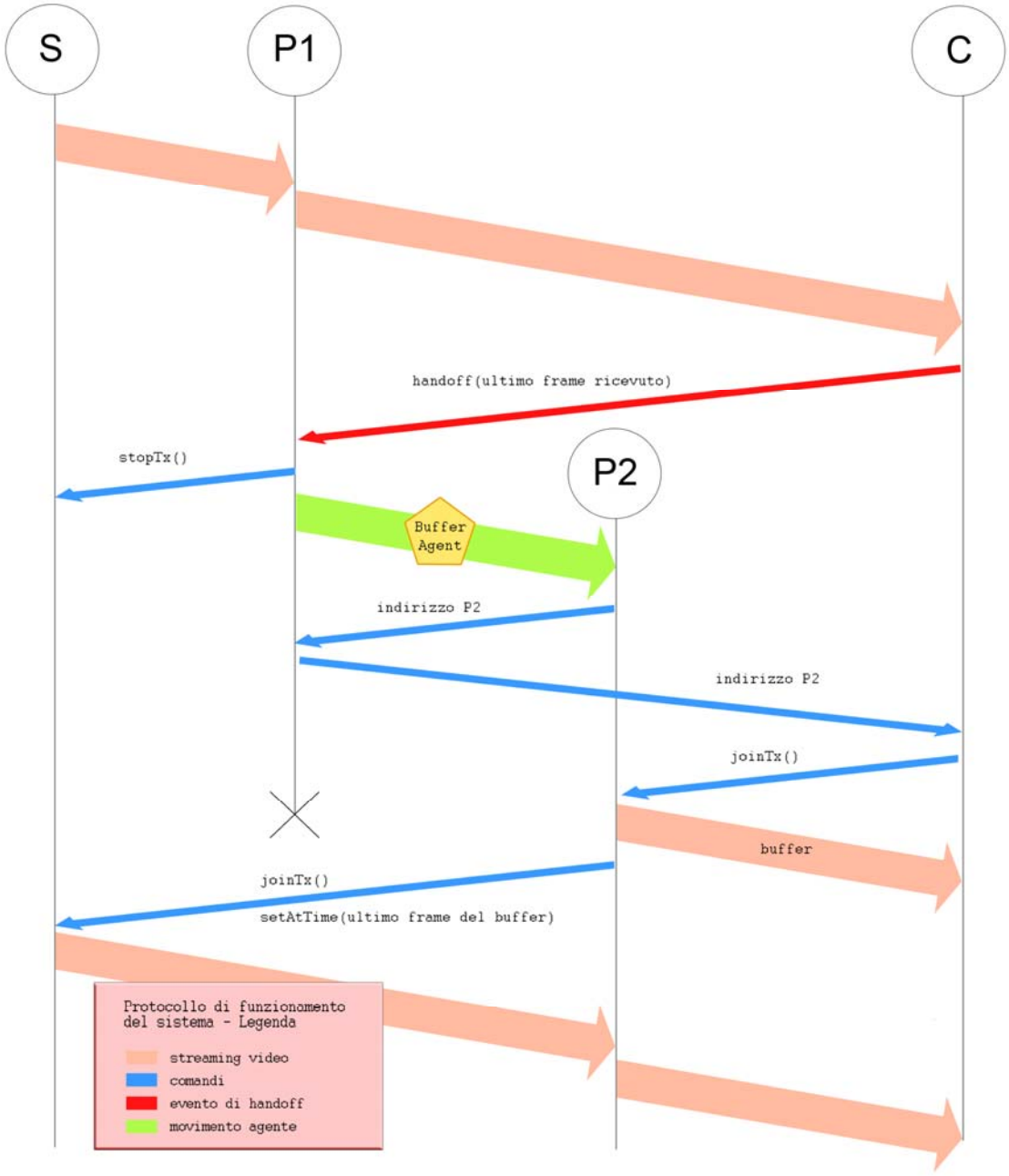
3.5 Integrazione con MUM

La struttura buffer da implementare dovrà essere integrata in MUM. Inoltre l'applicazione realizzata per spostare il proxy dovrà inserirsi nell'architettura multilivello dell'infrastruttura per mezzo di interfacce e utilizzare i servizi attraverso di esse. Per questo è necessario un attento studio del sistema.

Una complessità di MUM è il sistema utilizzato per caricare le classi. Attraverso il metodo *getComponentInstance* è possibile richiedere al Download Manager una determinata classe, indicando il package e l'interfaccia della classe stessa come stringa di caratteri. Trattandosi di un sistema distribuito questo è l'unico modo per recuperare risorse remote.
Questa soluzione però non permette nessun tipo di controllo automatizzato da parte dell'ambiente di sviluppo (es Eclipse) che non può fornire allo sviluppatore una visione globale dell'applicazione.

4. Progetto

4.1 Protocollo di funzionamento



Protocollo di funzionamento del proxy durante l'handoff.

Il protocollo di funzionamento del sistema è quello illustrato in questo diagramma di sequenza. Prendiamo in considerazione il sistema in funzionamento regolare: il server invia il flusso multimediale (freccia rosa) al proxy P1 che a sua volta lo inoltra al client. A un certo istante il client richiede lo spostamento del proxy provocando un handoff della connessione, come detto in precedenza nella realtà questo evento è provocato dallo spostamento del client stesso su un'altra cella ma nella nostra simulazione ci concentreremo soltanto sul funzionamento del proxy.

Ricevuto il comando, il proxy interrompe la trasmissione del flusso multimediale inviando al server il comando di interruzione dello streaming (stopTx), crea l'agente BufferAgent a cui delega l'installazione di un nuovo proxy (P2). L'agente accetta come parametri:

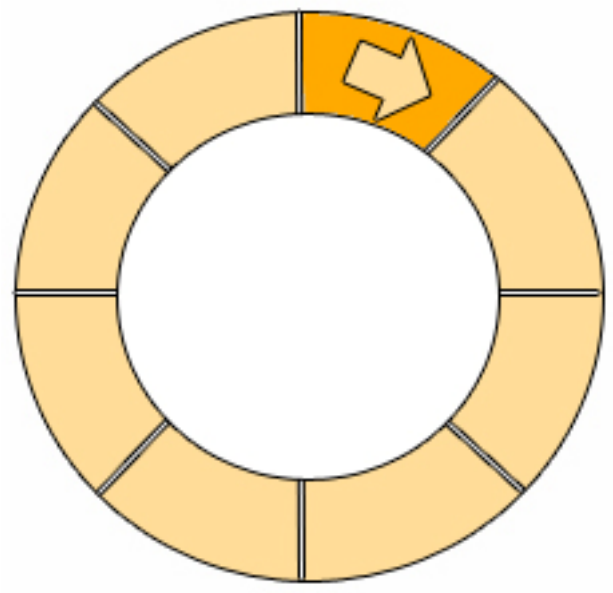
1. l'indirizzo del server al quale P2 si deve riconnettere al termine dell'installazione,
2. il buffer circolare contenente i frames non ancora inviati al client ,
3. il plan per raggiungere il nodo destinazione.

Una volta installato P2 il nuovo proxy notifica il suo indirizzo a P1 e richiede la connessione al server (joinTx). Intanto P1 invia l'indirizzo di P2 al client, che può richiedere a sua volta la connessione con P2 (joinTx con P2). P1 a questo punto può terminare. Quando P2 e il client sono stati connessi P2 comincia a fare streaming con i frames già contenuti nel buffer e, mentre questi vengono consumati, si riconnette al server richiedendo l'inizio dello streaming dal punto in cui si era interrotto il

caricamento del buffer. A questo punto il buffer continua a caricarsi con i frames ricevuti dal server.

4.2 Buffer circolare mobile

Al fine di ottimizzare e completare le funzionalità del proxy è necessario dotare questa entità di un buffer circolare mobile in grado di riattivarsi su un nuovo proxy. Un buffer circolare ha la caratteristica di memorizzare i frames in modo sequenziale in una struttura ad anello e di continuare a memorizzarli sovrascrivendo i frames già letti a partire dalla prima posizione disponibile.



Struttura del buffer circolare, la freccia nella posizione di partenza indica il verso di riempimento

Nel caso in esame infatti ci troviamo su un proxy che riceve un flusso multimediale. Tale flusso viene prima memorizzato sul buffer e poi inviato sulla rete. In questo modo, quando il cliente si sposta su un altro place e interrompe la comunicazione col proxy attuale, la parte di filmato che questo proxy ha ricevuto dal server non viene persa ma è inviata attraverso il buffer mobile sul nuovo proxy che andrà a servire il cliente migrato.

In particolare questo spostamento avverrà per mezzo dello stesso agente che ha il compito di installare il nuovo proxy sul place di destinazione.

4.3 Memorizzazione di un singolo frame

Come detto in precedenza gli oggetti che JMF mette a disposizione per memorizzare i singoli frames non sono serializzabili. E' necessario quindi ricreare queste API in modo da ottenere un tipo di dato trasformabile in una sequenza di bit.

A questo scopo è necessario anche creare metodi che compiono la trasformazione da ExtBuffer a un nuovo oggetto serializzabile e viceversa. Il proxy continua quindi a ricevere un DataSource da cui vengono estratti gli ExtFrames, che il buffer circolare accetta come parametri. La trasformazione in oggetti serializzabili avviene all'interno di questo prima di essere memorizzati e inviati sequenzialmente. La classe che rappresenta il singolo frame, per essere serializzabile, contiene solo attributi primitivi e oggetti Object.

4.4 Spostamento del buffer tramite agente

Gli agenti mobili permettono di spostare oggetti con stato e aggiungere codice dove necessario. In questo caso le entità da spostare sono il buffer e il proxy stesso che deve poi essere riattivato in modo da garantire la continuità della sessione.

L'agente deve essere programmato con un piano di migrazione, nel quale è indicato il percorso da effettuare per raggiungere il place obiettivo, come riconoscerlo e quali componenti installare per la sua messa in opera.

Un agente con queste caratteristiche è già presente in MUM. Il PlanVisitorAgent infatti ha il compito di inizializzare il sistema installando le entità che lo compongono con il codice necessario e inizializzarle attraverso altri agenti. Il nuovo agente addetto all'installazione del proxy e allo spostamento del buffer sarà ricalcato su quello esistente e fornito di un proprio piano di azione.

4.5 Componente Proxy

Per realizzare questo progetto è necessaria, oltre alla creazione, la modifica di alcuni componenti. Il server rimane invariato mentre è necessario un nuovo tipo di proxy che gestisce un buffer circolare e la modifica del Cliente.

Il proxy deve gestire una politica di bufferizzazione e una per il suo spostamento al termine del quale deve riconnettere al client.

Il proxy ha quattro stati. Nel primo il buffer inizia a caricarsi di frames, lo spostamento del proxy è possibile, ma poco conveniente in quanto la disponibilità di

bufferizzazione può essere limitata. Nel secondo, assumendo che i frames continuino ad arrivare sul proxy con una frequenza abbastanza costante, il buffer si mantiene quasi sempre pieno e continua ad essere caricato. L'accesso al buffer avviene secondo il modello produttore/consumatore: il buffer rappresenta il contenitore dove i dati vengono memorizzati fino al riempimento dello spazio, dopodichè il buffer blocca le operazioni di scrittura fino alla liberazione di posti per altri frames, in maniera concorrente un consumatore legge i frames liberando nuovo spazio.

Il terzo stato si raggiunge alla notifica della richiesta di spostamento, il buffer viene interrotto e iniziano i meccanismi di migrazione.

Nel quarto stato il proxy si trova sul nuovo nodo e riattiva il buffer, i frames non ancora consumati vengono trasmessi al client mentre gli spazi che si liberano accolgono i nuovi frames ricevuti dalla nuova connessione col server, quando questa è stata ristabilita.

Il proxy che ha inviato il BufferAgent per realizzare lo spostamento riceve una notifica quando l'installazione del nuovo proxy è stata completata e la invia al client che per tutta la durata dell'operazione è rimasto in attesa. Attraverso questa notifica il client riceve anche l'endpoint per la connessione al nuovo proxy. Una volta terminata questa fase il primo proxy libera le risorse sul suo place e termina l'esecuzione.

In MUM le ProtocolUnit gestiscono completamente la comunicazione tra i diversi endpoint attraverso connessioni TCP. Queste unità devono adattarsi al mutare degli endpoint. Perciò, nello scenario illustrato, quando il client riceve la notifica da

parte del nuovo proxy crea un protocollo di comunicazione con quest'ultimo, elimina il precedente, e notifica i nuovi dati al proxy che realizza a sua volta una ProtocolUnit.

4.6 Componente Client

Per simulare lo spostamento del proxy l'interfaccia grafica del client sarà dotata di un comando di tipo "Switch Proxy". Il comando inoltrato sul canale di controllo e ricevuto dal Proxy notifica la richiesta di spostamento di quest'ultimo. Il client si mette in attesa sospensiva durante la fase di spostamento del proxy.

Al termine di questa fase il client riceve un nuovo indirizzo per ristabilire la connessione col nuovo proxy e può iniziare a ricevere lo streaming.

5. Implementazione

5.1 Progetto dei componenti

Per realizzare un progetto di questo tipo è necessario innanzitutto disporre degli strumenti necessari alla memorizzazione del filmato. Occorre definire una struttura mobile che accoglie il singolo frame e i suoi metadati e un'altra più complessa che gestisce politiche di buffering.

5.2 Classe MoveableBuffer

Possiamo definire il frame l'unità fondamentale su cui lavora il buffer. La classe MoveableBuffer ricalca le funzionalità dell'ExtBuffer, cioè della rappresentazione del frame fornita da JMF, arricchendola della capacità di spostarsi come un flusso di bit sullo stream. La classe implementa a tale scopo l'interfaccia Serializable e contiene attributi primitivi (interi o booleani) e di tipo object (serializzabili). Il MoveableBuffer è composto da un costruttore:

```
public  MoveableBuffer(ExtBuffer buff)
```

Questo accetta in ingresso un ExtBuffer e ne copia gli attributi in quelli della classe stessa, nel caso di attributi complessi (Header, Data, Format) l'attributo viene copiato in una struttura object.

Per ottenere dal MoveableBuffer l'ExtBuffer originale è stato creato il metodo:

```
public ExtBuffer getBuffer()
```

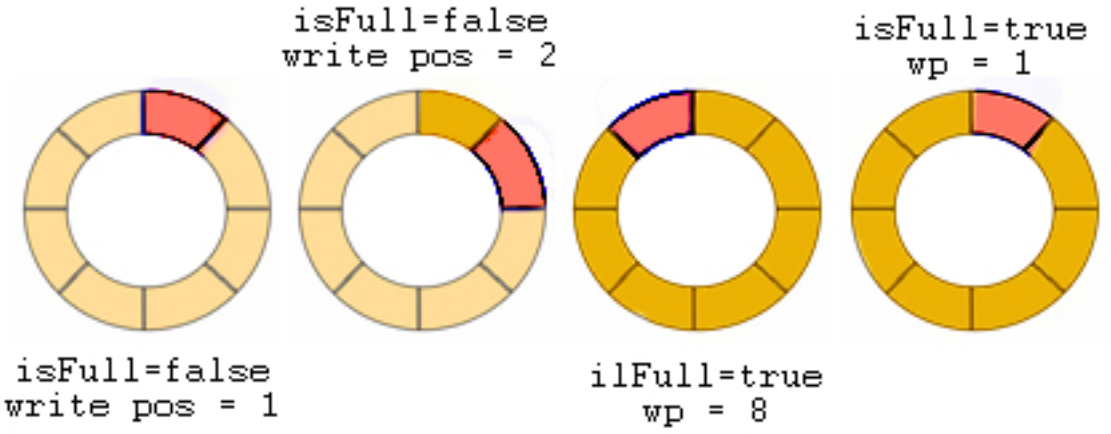
5.3 Classe MoveableCircularBuffer

Questa classe realizza il buffer e il suo coordinamento. Il buffer viene inizializzato con la dimensione desiderata, cioè impostato col numero massimo di frames che può contenere, e memorizza i frames ricevuti in un Array di MoveableBuffer. Il costruttore usato da questa classe è:

public MoveableCircularBuffer(int n)

Il buffer opera in modo sequenziale. Una volta occupata l'ultima posizione, nel caso nessun frame sia stato consumato, notifica lo stato di buffer pieno. Attraverso il metodo isFull() è possibile controllare lo stato di riempimento del buffer.

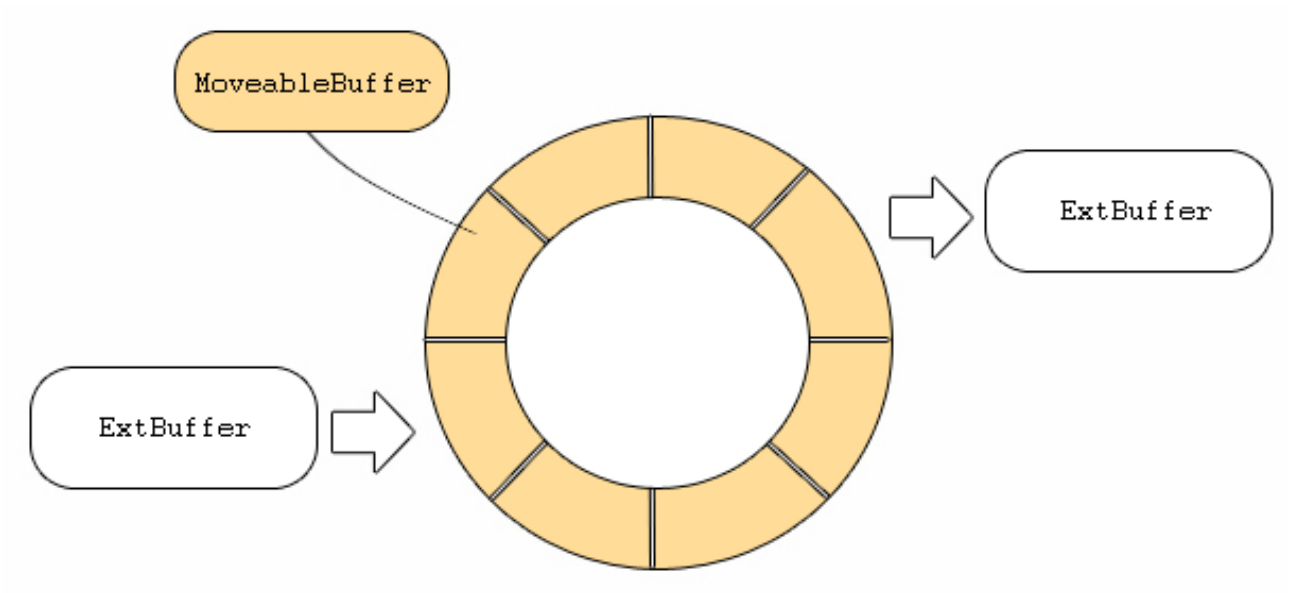
In condizione di funzionamento normale il buffer continua a caricarsi con i frames ottenuti dal data source e a rilasciarli sullo stream, per questo sono necessari strumenti di sincronizzazione tra la fase di scrittura e quella di lettura in modo che non vengano sovrascritti i frames non consumati.



Fase di scrittura del buffer con dimensione 8.

Per garantire la sincronizzazione in lettura e scrittura il buffer usa due puntatori: uno head che indica la posizione da cui è possibile leggere e uno tail, che punta alla posizione da cui è possibile scrivere. Inizialmente, quando il buffer è vuoto, head e tail puntano alla stessa posizione ma il metodo canWrite() blocca la lettura fino a quando non viene inserito il primo frame, questo metodo ritorna false ogni volta che head e tail puntano alla stessa cella di memoria.

Quando il buffer si trova sul proxy migrato la trasmissione dei frames riparte dall'ultimo frame ricevuto dal client. Il punto viene ritrovato attraverso il numero di sequenza che distingue ogni frame e il contenuto del buffer viene trasmesso fino all'ultima posizione occupata, memorizzata in una variabile che rappresenta lo stato interno del buffer.



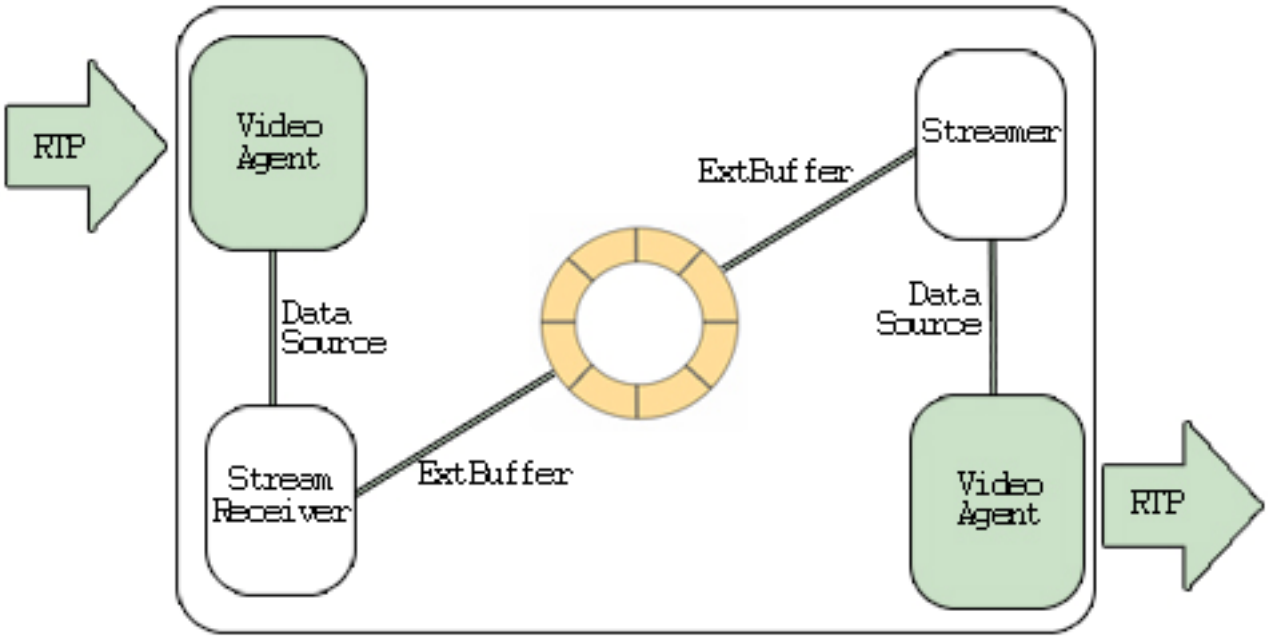
Il MoveableCircularBuffer, che utilizza oggetti serializzabile MoveableBuffer, ha in ingresso e in uscita ExtBuffer.

Il MoveableCircularBuffer riceve e restituisce i frames in strutture di tipo ExtBuffer (definite in JMF), la trasformazione da ExtBuffer in MoveableBuffer, formato in cui viene memorizzato il frame nel buffer, è perciò trasparente al programmatore che può usare le normali classi di JMF.

Anche la classe MoveableCircularBuffer implementa l'interfaccia serializable perchè deve potere essere spostato per mezzo di un agente.

5.4 Buffer Renderer, Streamer e StreamReceiver

Per caricare il buffer di frames il data source viene passato allo *StreamReceiver*, un componente che ha il compito di decodificare il flusso multimediale.



Catena di trasformazione del flusso RTP in un proxy.

Questo è un componente thread ed è mandato in esecuzione dopo essere stato inizializzato con il DataSource da leggere e il MoveableCircularBuffer su cui memorizzare i frames. Inoltre un flag permette di decidere se visualizzare il video quando il buffer è pieno per la prima volta.

Lo StreamReceiver crea un RawBufferParser() e vi associa il DataSource in ingresso. Quando il parser è stato attivato è possibile ricavare le tracce.

Ogni traccia rappresenta un canale di informazioni contenuto nel flusso, come l'audio e il video, ed è accessibile solo dopo un'operazione di demultiplexing.

Lo StreamReceiver continua a leggere una traccia, nel nostro caso video, e a memorizzare ogni frame sul buffer fino a alla lettura dell'EOM.

Lo Streamer ha la funzione inversa, si comporta cioè da multiplexer riunendo le tracce in un unico Data Source. Il multiplexer utilizzato è il H263RTPMultiplexer che trasforma i dati da formato H263 in RTP.

A livello di parser e multiplexer è possibile impostare il framerate del video in modo da regolare la velocità di lettura e scrittura del buffer.

Il BufferRenderer è il componente usato per visualizzare il contenuto del buffer ad esempio dal proxy e imposta la catena dei plugin necessari.

Per prima cosa imposta un decoder scegliendo tra JavaDecoder o SunDecoder, a seconda che la piattaforma su cui è installato il proxy supporti il formato nativo o meno, vi associa il Data Source, recupera i formati disponibili e imposta lo spazio

colori per i formati supportati. Infine crea un AWTRenderer da cui si ricava il componente visualizzabile su un pannello video.

5.5 BufferAgent

Il BufferAgent sposta il proxy e il buffer assieme ad esso. Viene lanciato dall'InitManagerAgent, l'agente che nel sistema presiede all'installazione dei componenti, attraverso il metodo *proxySwitch(Path, ComponentInfo, AgentID, MoveableCircularBuffer buffer).*

Questo metodo non è invocato direttamente dalla ProtocolUnit ma è stato aggiunto all'interfaccia IinitManager, e quindi alle sue implementazioni, per mantenere inalterato il concetto di centralizzazione delle operazioni di inizializzazione presente in MUM.

Per invocare questo metodo dalla ProtocolUnit è stato necessario aggiungere all'agentSystem (realizzato da SOMA) un metodo per ottenere direttamente un riferimento ad esso tramite factory, in questo modo:

```
AgentSystem as = ((IMUMFacilitiesProxySwitchEnabled)
factory).getAgentSystem();
```

Il BufferAgent è programmato con quattro argomenti:

1. id dell'agente creatore con cui il proxy può comunicare,
2. piano di spostamento da realizzare,
3. buffer circolare,

4. ComponentInfo dell’entità da cui riceve il flusso (server o client).

Il piano, ProxySwitchPlan, è dello stesso tipo usato per lo spostamento del client, ma è programmato da un metodo aggiunto al DecisionMaker con le informazioni necessarie allo spostamento e installazione di una entry di tipo proxy:

public Plan getPlanForProxySwitch(Path, ComponentInfo)

Il BufferAgent si sposta attraverso il path, in ogni nodo verifica se si trova sul target e in caso negativo prosegue il percorso lanciando il metodo *go(placeToGo, "run")*. Una volta raggiunto il nodo destinazione ottiene dall’agentSystem un downloadManager con cui esegue il download dei componenti elencati nel plan. In caso di insuccesso l’agente termina l’esecuzione.

Il risultato del download è notificato all’agente che implementa l’interfaccia *IswDownloadedListener*, attraverso i metodi downloadFinished(DownloadFinishedEvent) e downloadProblems(DownloadProblemsEvent) è possibile gestire gli eventi di successo o insuccesso di un download.

Una volta scaricati i componenti è possibile inizializzare il proxy e farlo partire.

```
IProxySessionManager proxy = (IProxySessionManager)
agentSystem.getComponentInstance(
                        entry.getEntityPackageName(),
                        entry.getEntityInterfaceName(),
                        parameters);
```

```
((IProxySessionManOnlyVideoProxySwitchEnabled)proxy).init
( agentSystem.getOut(), agentSystem, info,
entry.getInitializationParameters(), buffer);
```

Sempre attraverso l'agentSystem l'agente ottiene un'istanza dell'interfaccia *IProxySessionManager* e ne invoca il metodo init al quale passa come parametri:

1. un riferimento all'output,
2. un riferimento all'agentSystem che il proxy utilizzerà a sua volta per il recupero delle risorse,
3. le info del server a cui riconnettersi,
4. i parametri di inizializzazione relativi al componente (ricavati dal plan)
5. il buffer.

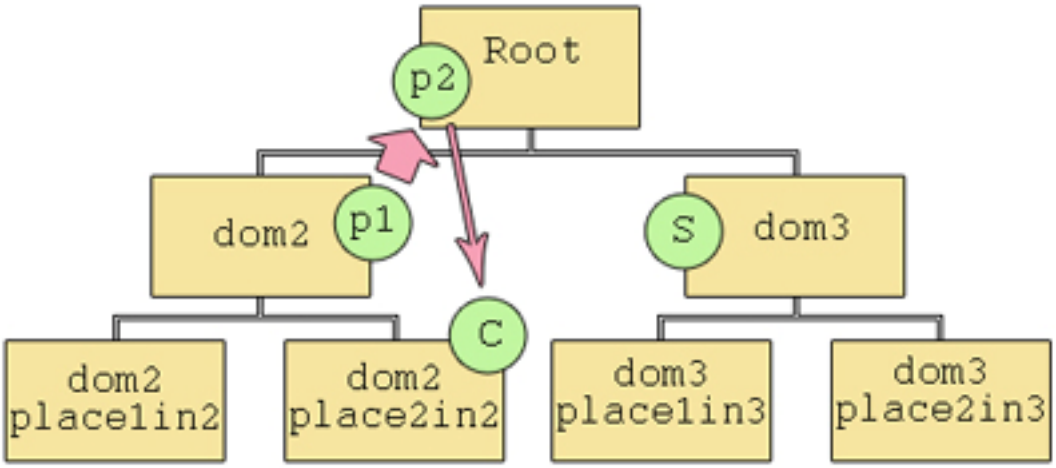
In seguito, con una procedura analoga, il buffer ottiene il riferimento e inizializza anche il cacheManager.

Infine il BufferAgent lancia il ProxyAgent per completare l'inizializzazione del cacheManager e gestisce la QoS.

5.6 Scenario della simulazione

Per la progettazione di un sistema bufferizzato in grado di gestire un evento di handoff del sistema ipotizziamo lo scenario illustrato nella figura sottostante. Il client

apre una sessione con il server per ricevere una risorsa di tipo multimediale. La comunicazione avviene attraverso un proxy intermedio. L'evento di handoff consiste nello spostamento di questo proxy su un altro place della rete SOMA da cui segue lo spostamento del buffer in esso contenuto.



Spostamento di un proxy su rete SOMA e riconnessione allo stesso client.

5.7 Funzionamento proxy

Prenderemo in esame l'implementazione che realizza lo scenario di simulazione descritto in precedenza e tutti i componenti e i meccanismi in esso coinvolti.

Il proxy, prima di ricevere il comando di spostamento, processa il dataSource attraverso il buffer circolare. Il momento ottimale in cui si vorrebbe muovere il proxy

è perciò quando il è buffer pieno. Il caricamento del buffer avviene a livello di InProtocolUnit, dove è possibile ricevere il DataSource.

Questo canale di controllo, usato dal VideoAgent, ha due metodi init invocabili dal livello di sessione: uno per l'attivazione da PlanVisitorAgent e l'altro per quella da Buffer agent. Quest'ultimo richiede come parametro il MoveableCircularBuffer.

Il VideoAgent, attraverso il canale di controllo del flusso verso il client (InProtocolUnit) realizza una socket che rimane in attesa dei comandi del client.

Quando il comando ricevuto è del tipo *joinStreaming* viene invocato il metodo *beginStreaming().* Nel comando sono incapsulate le informazioni relative alla presentazione che si vuole ottenere.

Il VideoAgent crea un altro canale di controllo per ricevere lo streaming dal server (OutProtocolUnit) che stabilisce una connessione RTP (lato server). A questo punto si attende l'evento che notifica l'avvenuta connessione e l'OutProtocolUnit può iniziare a ricevere il DataSource.

La InProtocolUnit distingue il caso in cui il buffer, dopo l'inizializzazione del canale stesso, sia vuoto o pieno, nel primo caso significa che il proxy non ha subito uno spostamento e perciò deve iniziare subito a bufferizzare il video ricevuto e farne streaming col client.

Il VideoAgent realizza il canale dati su cui avviene lo streaming. E' stato modificato per prendere in ingresso il buffer circolare che viene processato attraverso il multiplexer implementato dalla classe Streamer.

Quando la InProtocolUnit riceve il comando di SwitchProxy, contenuto nel package MUM.Messages, invoca il metodo do_switchProxy(). Questo comando incapsula l'oggetto clientAgentID:

```
clientAgentID=((SwitchProxy)cmd).getClientAgentID();
```

Il metodo do_switchProxy() delega l'operazione di spostamento all'InitManagerAgent fornendo il path da percorrere, le Info del server, l'ID del client e il buffer locale:

```
initManager.proxySwitch(path, info, clientAgentID, localBuffer);
```

Inoltre all'InitManagerAgent viene indicato come listener dell'operazione la stessa InProtocolUnit.
In questo modo, quando quest'agente ha lanciato il BufferAgent e l'istallazione del nuovo proxy è stata completata, il metodo initFinished recupera l'evento InitFinishedEvent con le ComponentInfo della nuova entità e le comunica al client.

```
public void initFinished(InitFinishedEvent ev)
```

Il buffer installato sul nuovo proxy contiene i frames non ancora spediti al client. Quando il proxy riceve il comando di beginStreaming ristabilisce la connessione al server con la stessa procedura descritta in precedenza ma inizia a fare streaming col contenuto del buffer remoto attivando lo Streamer.
Il buffer ricevuto può anche essere visualizzato direttamente sul proxy per mezzo del BufferRenderer.

5.8 Funzionamento cliente

Sul client, strutturato a livelli come il proxy, il comando di switch passa attraverso il protocollo di sessione e il video agent ed è inviato concretamente sul canale di controllo dalla protocol unit del cliente mentre i livelli sovrastanti attendono il termine dello spostamento del proxy. Il comando incapsula inoltre l'ID del cliente.
Il risultato, se l'operazione si è conclusa positivamente, sono le component Info del nuovo proxy.

E' il VideoClientSessionManager, il livello delegato alla gestione della sessione, a utilizzare queste informazioni per creare un nuovo video agent dopo avere distrutto il vecchia canale di controllo (protocol unit). In questo modo si disconnette dal vecchio proxy che può essere terminato.

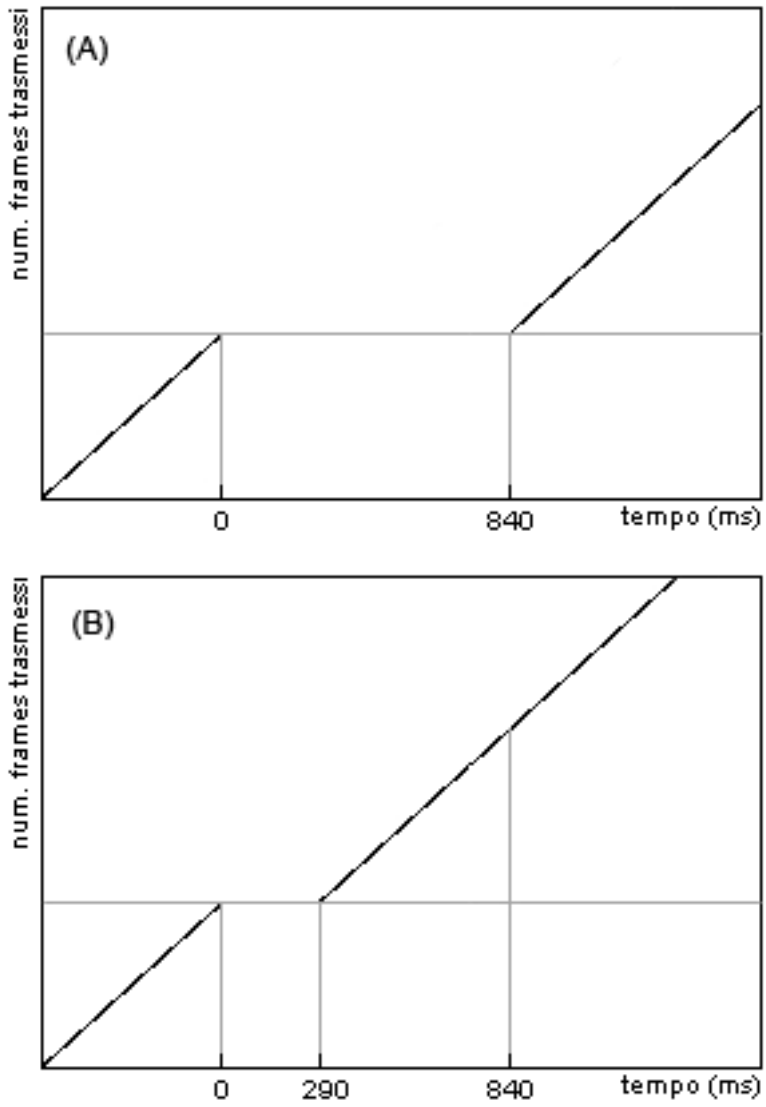
Il video agent e la protocol unit sono ricreati tramite factory accedendo alle risorse del repository.

Una volta lanciato, il VideoAgent inizializza a sua volta la nuova protocol unit che ha le informazioni necessarie a stabilire la connessione col nuovo proxy.

5.9 Simulazione

I componenti sono stati testati in locale sullo scenario illustrato in §5.6 su un computer PentiumIV 2Ghz. Il tempo che il BufferAgent impiega per raggiungere il nodo di destinazione (Root) dal nodo di partenza (place2, dom2in2) è praticamente trascurabile. Considerando una simulazione di tipo pessimista, dove il proxy viene spostato in un place che non contiene i componenti necessari al suo funzionamento, il BufferAgent impiega 10 ms per il loro download. Altri 30 ms sono necessari per attivare il proxy (cioè inizializzarne il SessionManager, il VideoAgent e i canali di controllo che stabiliscono le connessioni col server e il client). Il BufferAgent ha quindi una durata media di funzionamento di circa 40 ms. Se dopo l'handoff il proxy si trova su un place dove l'entità (proxy) è già stata installata (simulazione di tipo ottimista) il tempo di vita del BufferAgent si riduce a 30ms. La dimensione del buffer, testato con capacità diverse (200, 400 e 800 frames), non influisce sulla durata di queste operazioni.

Appena i componenti del proxy sono stati inizializzati dal BufferAgent, il nuovo proxy riattiva le connessioni al server e al client per realizzare lo streaming. Il tempo impiegato per ognuna di queste operazioni è nell'ordine dei 200-250 ms, mentre il data source viene ottenuto dopo 300 ms circa dall'inizializzazione del proxy.

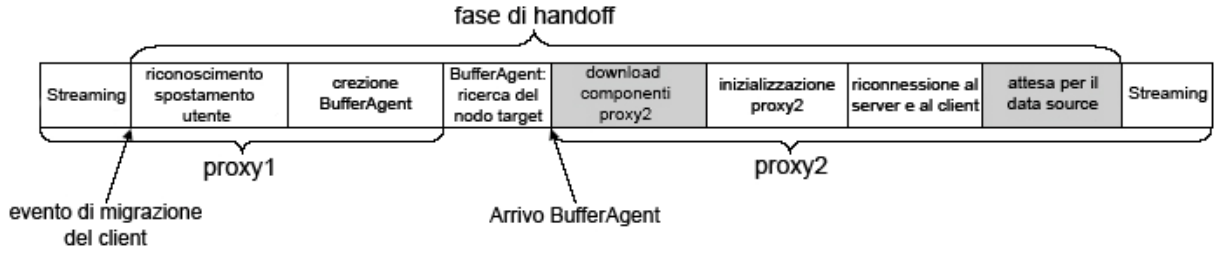


Tempo di riattivazione dello streaming da un proxy con buffer (fig.B) e senza (fig.A)

Ipotizzando di avere sul proxy un buffer adeguato a coprire i tempi di riconnessione al server e di arrivo del data source, lo streaming riprende dopo 290 ms dall'inizio dell'handoff.

Dalla parte del client la durata totale dell'handoff percepita è di circa 1,4 sec. Utilizzando il buffer circolare sul proxy, questa durata si riduce a circa 1,2 sec. Il buffer circolare sul client dovrà essere dimensionato in modo da colmare questo

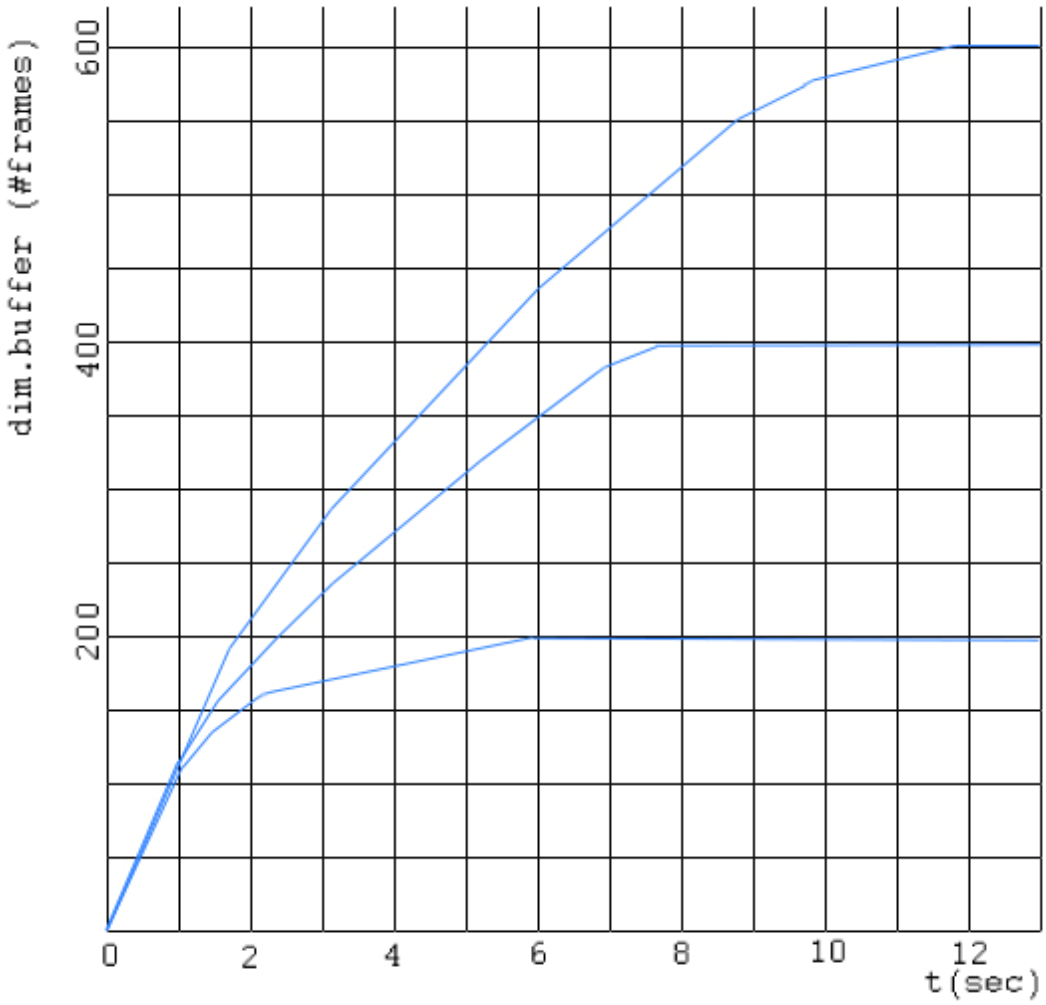
intervallo per evitare interruzioni nella visualizzazione. La figura sottostante rappresenta le fasi che compongono l'handoff ed evidenzia (in grigio) quelle di durata trascurabile in una simulazione con ipotesi ottimistiche e con proxy dotato di buffer.



Sequenza temporale delle fasi esaminate durante l'handoff

Un altro test è stato effettuato per monitorare il tempo impiegato dal buffer per caricarsi. Considerando come momento iniziale l'istante in cui il proxy riceve il data source, si nota come la memorizzazione dei frames avvenga rapidamente nei primi 2-3 secondi iniziali. Questo è il tempo impiegato dal client per connettersi al server e richiedere il data source. In questo intervallo il data source viene processato dal demultiplexer senza che alcun frame venga consumato.

Una volta iniziato lo streaming con il client, il multiplexer inizia a consumare i frames. La loro memorizzazione non ha i vincoli temporali della visualizzazione (framerate) per cui, come mostra il grafico, i frames vengono caricati più velocemente di quanto vengano consumati fino al riempimento del buffer. Il test è stato verificato con buffer di dimensione rispettivamente di 200, 400 e 600 frames.



Riempimento del buffer nel tempo

Conclusioni

Il lavoro realizzato contribuisce al miglioramento della continuità di un servizio di tipo VoD orientato alla rete wireless, con le problematiche di continuità del servizio e mantenimento della sessione che questo comporta. L'accesso alla rete attraverso terminali mobili è infatti soggetto alla perdita di connessione e all'immadiata riconnessione nel passaggio da una zona di copertura all'altra.

Questo problema (handoff) è stato arginato con la realizzazione di un proxy che porta il servizio 'vicino' all'utente. Il proxy è in grado di bufferizzare porzioni di filmato in modo da garantire una certa robustezza in caso di rallentamenti dello streaming dal lato del server; inoltre è in grado di muoversi da un nodo all'altro della rete senza perdere il contenuto del buffer, seguendo il movimento migratorio dell'utente.

Il miglioramento delle prestazioni si apprezza dal momento in cui, durante la visualizzazione di un contenuto multimediale, l'utente non percepisce il salto da una scena all'altra a fronte di una interruzione del servizio. Infatti nessun frame viene perso ed è possibile riattivare rapidamente lo streaming verso l'utente con il contenuto del buffer.

Il risultato è stato raggiunto con l'uso della tecnologia ad agenti mobili, sfruttando le capacità di dinamicità e mobilità che essa rappresenta.

Si prevede, per uno sviluppo futuro, l’implementazione di un buffer per ogni traccia del flusso, compresa quella audio, e la gestione dei problemi di sincronizzazione che questo rappresenta.

Un’altra possibile linea di sviluppo è quella di realizzare un buffer in grado di dimensionarsi a seconda delle necessità.

Bibliografia

[1] Yi Cui, Klara Nahrstedt, “Seamless User-level Handoff in Ubiquitous Multimedia Service Delivery”

[2] Alejandro Terrazas, John Ostuni, Michael Barlow: “Java Media APIs: Cross-Platform Imaging, Media, and Visualization”, 2001

[3] http://www.lia.deis.unibo.it/Staff/LucaFoschini/MUM/index.html “MUM: Mobile agent based Ubiquitous multimedia Middleware”

[4] http://www-lia.deis.unibo.it/Research/SOMA: “SOMA: Secure and Open “Mobile Agent”

[5] Gambaro e Ricciardi: “Economia dell’informazione e della Comunicazione”