Università degli Studi di Bologna

FACOLTÀ DI INGEGNERIA
Corso di Laurea in Ingegneria Informatica
Corso di Reti di Calcolatori

MIDDLEWARE PER SERVIZI MULTIMEDIALI ADATTATI VERSO DISPOSITIVI WIRELESS IN AMBIENTE J2ME

Tesi di Laurea di:
LUCA RENZI

Relatore:
Chiar.mo Prof. Ing. ANTONIO CORRADI

Correlatori:
Dott. Ing. PAOLO BELLAVISTA
Dott. Ing. LUCA FOSCHINI

Anno Accademico 2004-2005

Parole chiave:

J2ME,
Bluetooth,
Streaming Multimediale,
JMF,
MMAPI.

Indice

Introduzione

L'affermarsi di Internet ha messo a disposizione nuovi mezzi e strumenti per accedere a servizi e reperire informazioni. Il grande sviluppo si è avuto anche grazie al Web, che ha semplificato l'utilizzo di questa tecnologia suscitando grande attenzione da parte di tutti. Negli ultimi anni c'è stato un forte sviluppo delle tecnologie wireless e di dispositivi in grado di sfruttarne le potenzialità, come cellulari, Personal Digital Assistant, o in generale dispositivi mobili. Cambia il modo di accedere ai servizi: l'accesso può essere ottenuto con maggiore facilità, senza essere vincolati a trovarsi in un luogo particolare. Il dispositivo eroga il servizio anche durante lo spostamento con continuità, permettendo una maggiore libertà.
Questo nuovo contesto estende l'accesso alla rete con il risultato di ottenere una connettività più vasta e più facilmente fruibile. La popolarità di dispositivi mobili, come cellulari ha avuto un enorme successo, a livelli da eguagliare la telefonia fissa. Si sono affiancati altri dispositivi mobili con peculiarità sempre più vicine a personal computer, seppure con certe limitazioni di elaborazione e memorizzazione. Diventano quindi strumenti su cui è possibile installare applicazioni, collegarsi ad esempio Wi-Fi o Bluetooh e fruire di servizi multimediali.
Partendo da questi presupposti concentreremo il nostro lavoro su due concetti fondamentali: mobilità e continuità. La qualità del collegamento di un dispositivo mobile nel corso del suo spostamento è soggetta a frequenti variazioni. La connessione può essere disturbata, può interrompersi, ma se l'applicazione è robusta il servizio può in certe condizioni continuare. Lo scopo del nostro progetto è quello di garantire continuità nell'esecuzione di un flusso multimediale con un approccio innovativo rispetto allo stream multimediale tradizionale. Contenuti audio o video sono molto costosi da gestire e la loro riproduzione, soprattutto su dispositivi limitati come i cellulari, può richiedere lo scaricamento dell'intera presentazione prima di poterla riprodurre. Tecniche di streaming tradizionale ovviano al problema di scaricare per intero la presentazione dando la possibilità di eseguire la presentazione mentre la si sta scaricando, ma interruzioni della connessione possono bloccarla. Il

lavoro di questa tesi affronta questa problematica: vogliamo permettere mobilità e continuità del servizio adottando una soluzione che sia in grado di superare momenti di assenza di connettività e nel caso di prolungate disconnessione dalla rete decidere da che punto della presentazione ripartire.

Il capitolo 1 conterrà un'introduzione degli argomenti che stanno alla base della materia trattata: concetto di streaming multimediale, soluzioni proposte, l'approccio da noi proposto.
Il capitolo 2 descriverà meglio la tecnologia Bluetooth, da noi utilizzata nella realizzazione dell'applicazione, analizzando i protocolli e le modalità di connessione e i tentativi suggeriti per lo streaming multimediale.
Nel capitolo 3 verrà analizzata l'architettura del nostro sistema, andando a dettagliare i blocchi funzionali lato server e lato client.
Nel capitolo 4 si tratteranno gli strumenti che permettono di lavorare su oggetti multimediali, confrontandoli tra loro.
Nel capitolo 5 questi strumenti verranno impiegati e si procederà alla descrizione dell'implementazione dell'architettura descritta nel capitolo 3.
Nel capitolo 6 verranno elencati gli strumenti che utilizzeremo, sia in fase di sviluppo che di test su dispositivi reali, insieme ad un'analisi critica dei risultati ottenuti.

CAPITOLO 1

Streaming Multimediale in reti Wireless

Negli ultimi anni abbiamo assistito a un forte cambiamento e innovazione nel campo dei dispositivi portatili, dai personal digital assistant (PDA) ai cellulari, che ha permesso di realizzare applicazioni prima impensabili: grazie alle loro caratteristiche di elaborazione e di connettività è possibile fruire di tutta una serie si servizi affidandosi ad un collegamento wireless verso uno scenario di apertura come Internet . Molte applicazioni fanno affidamento su una alta velocità di comunicazione per servizi multimediali su web.

Le applicazioni multimediali per la loro natura richiedono una notevole impiego di risorse, molto spesso con tempi di risposta stringenti che garantiscano continuità. Se ad esempio stiamo inviando una presentazione su una stampante di rete, problemi di congestione potrebbero interrompere momentaneamente la stampa, per una ripresa successiva senza per questo pregiudicare l'esito del risultato. Diversamente non è accettabile ascoltare un brano musicale con interruzioni. Per caratterizzare la qualità del servizio vengono utilizzati caratteristiche come ampiezza di banda, throughput, tempestività nella risposta, affidabilità.

In questo capitolo introdurremo alcune problematiche legate allo streaming multimediale in generale e ci concentreremo poi sullo streaming verso dispositivi wireless. Vedremo che cosa si intende per streaming e le entità che vengono coinvolte. Ci sposteremo poi sul lavoro che è stato fatto in questa tesi proponendo un approccio innovativo per il download e la riproduzione di streaming audio. Nell'ottica di garantire una certa qualità del servizio introdurremo l'idea di sessione. Dopo avere discusso di come venga realizzato uno streaming tradizionale, le motivazioni e i vantaggi che derivano da questa tecnica di trasmissione, focalizzeremo il discorso sul quello che è stato fatto in questa tesi confrontando le due soluzioni.

1.1 Introduzione allo Streaming

Con il termine *streaming* si intende il modo di trasferire flussi di dati digitali con caratteristiche di soft real-time per permettere al destinatario di vederne il contenuto in maniera continuativa mentre riceve i dati.
Il contenuto può essere qualunque. Il vantaggio di utilizzare lo streaming è che rende possibile al destinatario la riproduzione del contenuto immediatamente, senza dovere aspettare di avere scaricata e memorizzata tutta la presentazione. Lo svantaggio è che la qualità sarà fortemente condizionata da molteplici fattori.
Uno *stream* è un flusso di pacchetti con contenuto multimediale. I pacchetti generalmente sono generati da uno *streaming media server* che legge i dati da una qualunque sorgente, che potrebbe essere un filmato memorizzato sul server o catturato dal vivo (ad es. da una webcam, microfono, televisione, ecc..). Lo streaming di contenuti già memorizzati prende il nome di *on-demand streaming*, mentre la cattura dal vivo è detta *live streaming*.
Il contenuto di solito è compresso usando un *codec* opportuno, che riduce la dimensione dei dati da trasmettere dal momento che i contenuti multimediali sono molto voluminosi. La velocità di trasmissione potrebbe non risultare sufficientemente alta a garantire il flusso. I contenuti audio e video possono essere compressi ottenendo una buona compressione (anche fino a 50:1) con una modesta perdita di qualità. Il bit rate dello stream specifica la quantità di dati compressi inviati nell'unità di tempo. Il bit rate normalmente viene misurato in bit al secondo. Il bit rate può essere costante o variabile. Il bit rate e il codec usato condizionano in maniera molto marcata la qualità del contenuto.
In uno scenario di streaming Internet i pacchetti generati sono inviati al destinatario tramite una rete a commutazione di pacchetto utilizzando un protocollo di *streaming*. Il destinatario utilizza un player per lo stream ricevuto che riceve i pacchetti, li decodifica con l'appropriato codec e riproduce il contenuto all'utente.
Lo streaming è sensibile ad errori e ritardi in trasmissione dal momento che è richiesta la continuità del flusso per una continuità della presentazione. Se alcuni pacchetti sono persi o ritardati durante la

trasmissione, il player potrebbe non essere in grado di decodificare correttamente i dati ricevuti e si potrebbero verificare errori ed interruzioni. Per compensare possibili ritardi streaming media player di solito ricevono una certa quantità di dati prima di iniziare la riproduzione operando il *buffering* del flusso entrante. Si ricorre al buffering anche quando la trasmissione peggiora a causa del peggioramento delle condizioni di rete. Il lavoro di questa tesi si concentrerà su streaming audio.

1.1.1 Mobile Streaming

Il termine *mobile streaming* è usato per indicare l'erogazione di flussi multimediali verso terminali mobili attraverso una connessione senza fili (wireless) . Un terminale mobile può essere un cellulare o un PDA, fornito di supporto IP e software per la riproduzione. Come mostrato in figura 1.1 un dispositivo mobile può accedere alla rete con un collegamento wireless come Bluetooth o Wi-Fi

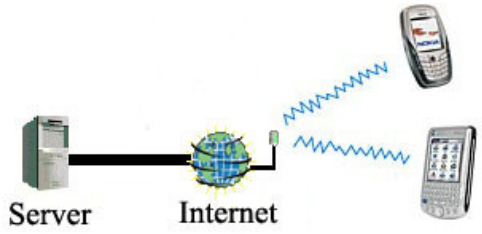


Figura 1.1 - Connessione Bluetooth

In seguito faremo riferimento al collegamento Bluetooth, che seppur presenta una velocità più bassa rispetto al Wi-Fi, è disponibile nella maggior parte di dispositivi mobili di ultima generazione. Rispetto a tecnologie come GPRS o UMTS, Bluetooth e Wi-Fi utilizzano una banda di frequenze libere.

1.1.2 Cambiamenti introdotti dal Mobile Streaming

A seconda della tecnologia wireless utilizzata abbiamo diverse velocità di trasmissione, dai 9,6 Kbps del GSM, ai 115,2 Kbps del GPRS agli attuali 384 Kbps dell'UMTS.

La limitazione di banda cui dispositivi mobili sono solitamente soggetti è un fattore che condiziona molto le scelte progettuali.
Altre tecnologie come Bluetooth e IEEE 802.11b arrivano rispettivamente a 723 Kbps e 11Mbps.
Per ridurre in ogni caso la mole di dati per contenuti multimediali è necessario utilizzare codec opportuni per la compressione. Sfortunatamente algoritmi di compressione molto complessi richiedono più computazione per la decodifica e ciò non è sempre è accettabile per dispositivi mobili che hanno prestazioni limitate. Potrebbe essere richiesto un utilizzo eccessivo del processore, che potrebbe portare ad un uso eccessivo della batteria e problemi di surriscaldamento.
La limitazione di banda non è poi l'unico problema in cui si può incorrere nelle reti wireless. Tutte le reti wireless sono soggette a continue variazioni di prestazioni.
Esistono molti di elementi che possono incidere sulla qualità di trasmissione:

- le condizioni atmosferiche,
- le variazioni di potenza del segnale causate dallo spostamento in luoghi dove le trasmissioni radio arrivano con difficoltà;
- lo spostamento da una zona coperta da un access point ad un'altra con diverso access point;
- la batteria, che determina più o meno potenza in ricezione e trasmissione;

sono solo alcuni esempi.
Se la qualità del canale peggiora, sono necessari più bit per la correzione degli errori.
L'ampiezza di banda può ridursi anche se la stessa risorsa è condivisa tra più utenti. Ci sono poi limitazioni dovute alla durata della batterie e alle ridotte dimensioni dello schermo. Elaborazioni troppo pesanti e continuate portano ad utilizzo intensivo del processore, ciò può portare a un surriscaldamento del dispositivo e a un calo della durata della batteria.
In questi dispositivi le funzionalità di input e output sono molto limitate. Per le modalità di input basti pensare ad una tipica tastiera di un cellulare con le frecce di direzione, un bottone di selezione e due bottoni.

Per l'output come vedremo più avanti le specifiche MIDP richiedono che un dispositivo abbia uno schermo con almeno 2 colori e una risoluzione di 96x54 pixel.

1.1.3 Standards per Mobile Streaming

Sono state sviluppate molte tecnologie di streaming proprietarie, che utilizzano caratteristiche di compressione e trasmissione diverse del contenuto multimediale.
Questo diventa un problema se non esiste interoperabilità tra le stesse. Sui dispositivi mobili questo è uno dei problemi maggiori, dove ancora oggi molte delle caratteristiche supportate dipendono dal tipo di dispositivo. Per garantire comunque un ambiente meno vincolante sono stati proposti alcuni standard, tra cui ricordiamo 3GPP PSS (Third Generation Partnership Project Packet-Switched Streaming,) e MPEG-4 (Moving Picture Experts Group).
3GPP PSS non specifica la codifica da usare o il contenuto dei dati, ma utilizza codec, formati e tipi di dati già standardizzati [3GPP05]. Per la codifica audio sono utilizzati i codec AMR e MPEG-4 AAC, per la codifica video i codec H.263 e MPEG-4.
3GPP PSS è supportato dalla maggioranza di piattaforme di streaming quali RealNetworks, Apple e Packet Video. E' implementato anche nella maggioranza di cellulari. Questo fa del 3GPP il più importante standard per mobile streaming al giorno d'oggi.

1.1.3.1 MPEG-4 Standard

MPEG-4 è uno standard di codifica per presentazioni audio-video sviluppato dal Moving Picture Expert Group (MPEG) [MPEG05] rilasciato nel 1999. Lo standard MPEG-4 (ISO/IEC14496) inizialmente fu indirizzato per compressione audio e video con un basso utilizzo di banda, ma durante il suo sviluppo è stato esteso diventando uno standard per applicazioni multimediali.
Lo standard MPEG4 introduce numerosi profili e livelli per permettere una configurazione precisa e granulare del processo di codifica. Profili e livelli assicurano che diverse implementazioni siano compatibili tra loro

permettendo a questo standard di essere utilizzato in applicazioni molto diverse e con scopi diversi.
L'utilizzo di profili e livelli per audio e video, permette al media di essere adattato per un certo dispositivo e per una certo tipo di rete e si adatta bene ad essere utilizzato per lo streaming in ambiente mobile wireless.

1.1.4 Piattaforme per Mobile Streaming

Molte aziende che sviluppano tecnologie per lo streaming multimediale si sono indirizzate verso tecnologie standard, ciò è molto importante, soprattutto nello streaming verso dispositivi mobili dal momento che difficilmente possono supportare le tecnologie proprietarie più disparate.
Tra le principali piattaforme che sono diventate standard troviamo:

- Real Networks Helix Universal Server: è un server per contenuti multimediali che supporta live e on-demand streaming con i principali formati.
- Apple QuickTime server: è un server per piattaforme Mac Os X
- Darwin Streaming Server: è una versione open source del precedente, disponibile anche per altre piattaforme.
- Packet Video pvServer: server per streaming rivolto principalmente a dispositivi 3G.

1.2 Mobilità, Handoff e Continuità di Servizio

Ci sono diversi problemi che sorgono quando si deve garantire continuità di trasmissione verso un dispositivo wireless. La variazione della velocità della connessione e influenzata da molti fattori e con lo spostamento del dispositivo si può raggiungere un'area meglio coperta da un altro access point. Questo passaggio, indicato con il termine *handoff* è molto importante e delicato e richiede molti accorgimenti per renderlo più indolore possibile. Cambiano i dispositivi a cui un utente mobile è collegato, ma il servizio deve continuare. A livello utente il passaggio ad access-point diversi risulta trasparente.
In un flusso tradizionale, lo streaming è continuo. Possiamo pensare ad uno scenario unicast (uno a uno) o multicast (uno a molti), dove esiste la possibilità di adattare la qualità e quindi la quantità di dati trasmessi in

relazione alle caratteristiche di disponibilità di banda. Se la rete ha notevoli ritardi, problemi di congestione lo stream può adattarsi alle sopraggiunte condizioni per garantire la continuità del servizio. E' ovviamente necessario un meccanismo di feedback.
Nella tesi verrà proposto un approccio innovativo, che a fronte di handoff, cadute di connessione, degrado delle prestazioni di rete è comunque in grado di continuare la riproduzione del contenuto multimediale, in particolare lavoreremo su un flusso audio.
Questo sarà reso possibile dal fatto che il dispositivo avrà già memorizzato la quantità di presentazione che verrà poi eseguita quando la connettività verrà meno. Al successivo ripristino della comunicazione il dispositivo riprenderà ad acquisire il media. Mentre vengono eseguiti questi passaggi l'utente continua a ricevere la presentazione senza interruzioni.
A seconda della tecnologia di trasmissione utilizzata può essere impiegata una compressione opportuna, tenendo conto ovviamente delle caratteristiche del dispositivo mobile.

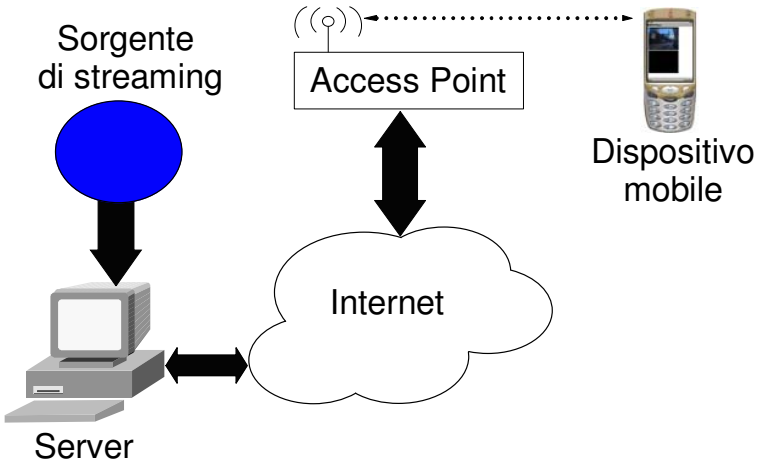


Figura 1.2 - Schema di funzionamento

Rispetto allo streaming tradizionale, l'operazione di buffering proposta in questa tesi si differenzia per granularità.
In uno streaming tradizionale troviamo le entità client-server: il server prepara lo streaming, e utilizza un buffer da cui il client può attingere attraverso il collegamento wireless. Il client utilizzerà un proprio buffer per la gestione del flusso.

Nella nostra proposta lato server il flusso verrà spezzato in tante clip della durata di qualche secondo che il client potrà scaricare e riprodurre. Il client scaricherà una quantità sufficiente di clip tali a garantire una continuità della presentazione anche quando ci saranno interruzioni di trasmissione. Verrà utilizzato il per la comunicazione il protocollo HTTP, che permette di lavorare ad un livello che non ci vincola a dettagli dipendenti dai dispositivi.
Il server a fronte di più richieste da parte di altrettanti client li informa di come ottenere le risorse necessarie per iniziare la riproduzione.
Ogni client gestirà la propria *sessione* e deciderà quando iniziare la riproduzione e decidere le modalità di download della presentazione in relazione alle condizioni di collegamento.
Il client potrà a fronte di problemi di connessione decidere quanta presentazione scaricare, decidere se dopo una caduta di connessione prolungata, riprendere da dove si era interrotta la comunicazione o da un altro punto, anche in riferimento al contenuto del media.

1.2.1 Valutazioni

Streaming e collegamento wireless, sono due concetti che presi singolarmente necessitano di tecniche molto complesse per raggiungere un grado di affidabilità che permetta il loro utilizzo. Negli ultimi anni sono stati fatti grandi passi e si ritrovano molte applicazioni che ne fanno uso. Realizzare uno streaming che utilizza un collegamento wireless richiede un'attenzione ancora maggiore. Ci devono essere una forte tolleranza ai guasti, errori di trasmissione e variazioni delle condizioni in cui si opera. Nella nostra architettura il server svolge un ruolo importante. Il server riceve uno stream e si occupa della sua compressione e frammentazione. Frammentare lo stream multimediale ha diversi vantaggi: interruzioni di breve durata non condizionano il servizio. Possiamo usare il protocollo HTTP per la comunicazione tra server e dispositivo mobile. Lavoriamo quindi ad un livello molto alto senza entrare nei dettagli implementativi di particolari dispositivi. Questo protocollo è disponibile nella stragrande maggioranza delle implementazioni dei dispositivi.
Il collegamento wireless che utilizzeremo è il Bluetooth, dal momento che è molto più facile da trovare implementato nella maggior parte di

dispositivi mobili. Il Wi-Fi sarebbe più veloce, pur operando nella stessa banda di frequenze, ma lo si trova più su PDA che su cellulari, per cui la nostra scelta è caduta sul Bluetooth.

1.2.2 Problematiche

Nella riproduzione di contenuti multimediali, è necessario attendere un certo tempo prima di potere eseguire la riproduzione. Nello stream tradizionale, la sorgente viene catturata ed elaborata e successivamente inviata al dispositivo che la dovrà riprodurre. Tutto questo richiede tempo e comunque, anche se di poco la trasmissione risulta differita. La trasmissione di streaming di una radio su internet può avere diversi secondi di ritardo, ma non necessariamente rappresenta un problema.
Nella nostra soluzione ci sono gli stessi problemi di ritardo. Va detto però che lato server viene fatta una frammentazione discreta del flusso realizzando clip della durata di qualche secondo.
Per garantire una certa fluidità della riproduzione è necessario attendere, prima di iniziare la riproduzione, l'acquisizione di almeno una clip. In generale a seconda del ritardo rispetto al server e delle condizioni di connessione e delle capacità di memorizzazione del dispositivo possiamo anticipare il download a 2, 3 o più clip. Garantendo una robustezza nella continuità in rapporto alla durata complessiva della presentazione già acquisita e non riprodotta.

1.4 Conclusione

In questo capitolo abbiamo parlato di streaming multimediale, orientato ad una connessione wireless. Tra le connessioni wireless esistenti concentreremo la nostra attenzione verso collegamenti Bluetooth, disponibili in molti dispositivi di nuova generazione che si trovano sul mercato. Abbiamo introdotto l'idea di cosa verrà introdotto di nuovo in questa tesi per garantire a fronte di handoff o in generale di caduta di connessione una continuità nella riproduzione, risulterà importante il concetto di sessione che determinerà le azioni da intraprendere al verificarsi di problemi, decidendo da dove riprendere la presentazione.
Per garantire la massima portabilità lavoreremo senza fare riferimento ad hardware particolari, in questo senso Java fornisce un grande contributo, e

utilizzeremo protocolli pienamente supportati dalla maggior parte dei dispositivi, come l'HTTP.

CAPITOLO 2

Related works

In questo capitolo parleremo di tecnologia Bluetooth, del suo funzionamento e delle tecniche adottate per limitare i periodi di disconnessione quando un dispositivo si sposta da un access point ad un altro (handoff).
Vedremo come i dispositivi possono tra loro collegarsi e sempre in ambiente Bluetooth le varie configurazioni di rete per il supporto alla mobilità, facendo anche riferimento alle reti LAN e servizi IP, ponendo una particolare attenzione alla QoS e sempre nell'ottica della qualità vedremo una tecnica di adattamento per applicazioni video streaming.

2.1 Panoramica sulla tecnologia Bluetooth

Bluetooth [BTM05, BTS05] è uno standard per comunicazioni radio wireless a corto raggio, bassa potenza e basso costo. Nasce nel 1994 sviluppato da Ericsson per rimpiazzare il cablaggio, e oggi disponibile in molti tipi di dispositivi. Per fare alcuni esempi possiamo elencare PDA, cellulari, PC, mouse, tastiere, joystick, macchine fotografiche, penne digitali, stampanti, access point LAN, auricolari, casse.
Nel 1998 Ericsson ha unito i propri sforzi con Intel , IBM, Nokia e Toshiba per formare il Bluetooth Special Interest Group (SIG). Nel 1999 si sono aggiunte 3Com, Lucent/Agere e Microsoft, garantendo al Bluetooth di svilupparsi come standard aperto assicurandogli un'accettazione rapida e piena compatibilità con il mercato. Le specifiche Bluetooth si trovano in [BT2005]. Oltre 2100 aziende lo supportano. La Wireless Personal Area Network (WPAN), tecnologia basata sulle specifiche Bluetooth, è uno standard IEEE che prende il nome di 802.15 WPANs.
Le specifiche Bluetooth definiscono come i dispositivi devono lavorare in gruppo. Una Bluetooth Wireless Personal Area Network (BT-WPAN) è composta da *piconet*. Ciascuna piconet è un gruppo di dispositivi (fino a 8).

Un dispositivo è denotato come master, e gli altri slave. Due piconet possono essere collegate tra loro tramite un dispositivo Bluetooth (gateway o bridge) per formare una *scatternet*.

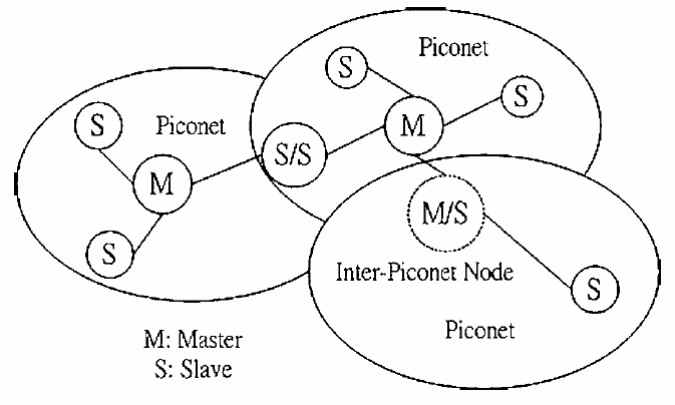


Figura 2.1 - Connessioni inter-piconet in scatternet

Le piconet interconnesse all'interno della scatternet formano la struttura portante per la Mobile Area Network (MANET), e permette a dispositivi che non comunicano direttamente o che sono fuori dalla loro portata di scambiarsi dati tramite diversi passi nella scatternet. Le attuali implementazioni di Bluetooth dipendono principalmente su semplici collegamenti punto a punto tra i dispositivi che sono nel raggio di copertura dell'altro. Le specifiche Bluetooth permettono anche soluzioni di connessioni e topologie più complesse. E' possibile formare Bluetooth scatternet che realizzano una comunicazione efficiente in più passi con risposte e consumi accettabili per soluzioni end-to-end.

2.1.1 Lo stack di protocolli Bluetooth

Le specifiche Bluetooth dividono lo stack di protocolli Bluetooth in tre gruppi [DEP05]:

- Transport Protocol group;
- Middleware Protocol group;
- Application group.

Il Transport Protocol group permette ai dispositivi Bluetooth di individuare altri dispositivi Bluetooth e di gestire la connessione a livello fisico e logico con i protocolli e applicazioni soprastanti.

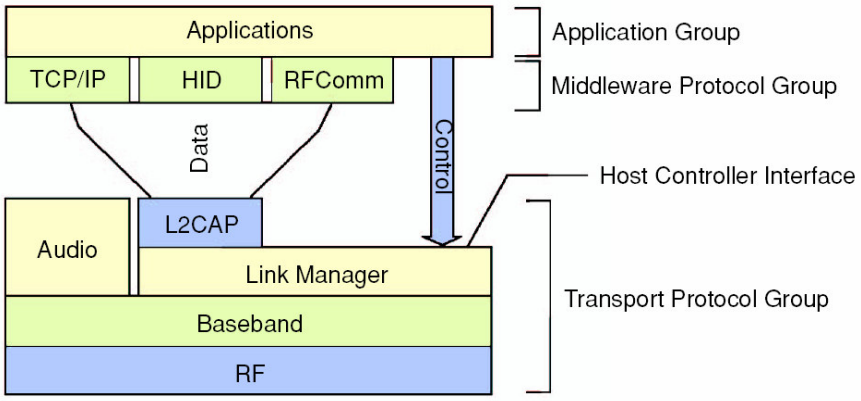


Figura 2.2 - Bluetooth Core Protocol Group

E' da notare che l'uso della parola "transport" nel Trasport Protocol group non indica la coincidenza con il livello di trasporto nel modello di riferimento OSI, piuttosto corrisponde al livello di collegamento e fisico.
I livelli Radio, Baseband, Link Manager, Logical Link Control and Adaptation (L2CAP) e Host Controller Interface (HCI) sono raggruppati nel Transport Protocol group.
Questi protocolli supportano sia la trasmissione sincrona che asincrona. Tutti i protocolli in questo gruppo sono necessari alla comunicazione tra dispositivi Bluetooth.
Il Middleware Protocol group include protocolli di terze parti e standard industriali. Questi protocolli permettono ad applicazioni esistenti e a nuove applicazioni di operare su collegamenti Bluetooth. Protocolli standard industriali comprendono i protocolli Point-to-Point Protocol (PPP), Internet Protocol (IP), Transmission Control Protocol (TCP), wireless application protocol (WAP), e Object Exchange (OBEX), adottati da Infrared Data Association (IrDA).
I protocolli Bluetooth sviluppati da SIG includono:

- Emulatore di porta seriale (RFCOMM) che permette ad applicazioni legacy di operare sul protocollo di trasporto Bluetooth;
- Protocollo basato su pacchetti Telephony Control Signaling (TCS) per gestire operazioni di telefonia.
- Protocollo di Service Discovery (SDP) che permette a un dispositivo di ottenere informazioni sui servizi disponibili sugli altri dispositivi.

E stato sempre prioritario sviluppare le specifiche Bluetooth riutilizzando protocolli già esistenti, come mostrato in figura 2.3.

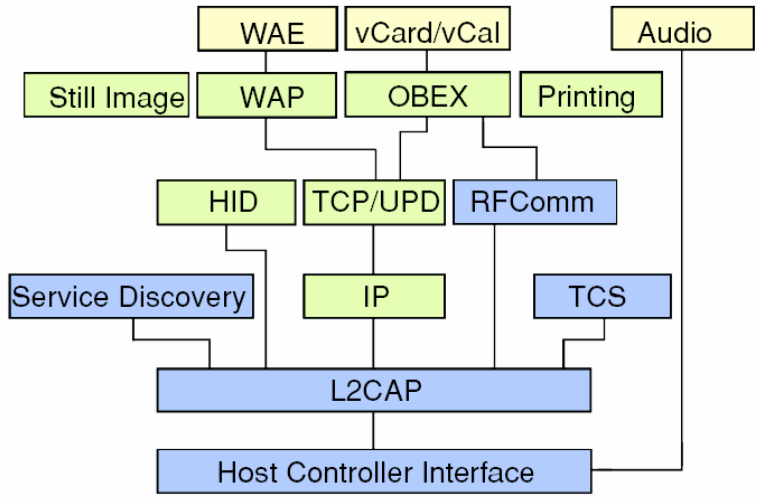


Figura 2.3 - Interoperabilità con protocolli ed applicazioni esistenti

L'Application group comprende applicazioni che utilizzano collegamenti Bluetooth. Possono essere applicazioni legacy sia applicazioni realizzate nell'ottica di essere utilizzate con Bluetooth.
Vediamo meglio i livelli nel Transport group:

- *Livello Radio*: Le specifiche di questo livello riguardano principalmente le modalità fisiche di trasmissione, come vedremo meglio più avanti;
- *Livello Baseband*: Questo livello definisce come un dispositivo Bluetooth cerca e si collega ad altri dispositivi. I ruoli master e slave che un dispositivo può assumere vengono definiti qui, come il passo di frequenze e la sequenza da utilizzare. Il dispositivo usa per interfacciarsi il time division duplexing (TDD), schema di polling basato su pacchetti. Master e slave comunicano ciascuno nello slot di tempo concordato. Inoltre a questo livello sono definiti i tipi di pacchetti, le procedure di processazione dei pacchetti e le strategie per il rilevamento e correzione degli errori, sbalzi di segnale, cifratura, trasmissione e ritrasmissione di pacchetti. Il livello Baseband supporta due tipi di collegamenti: Synchronous Connection Oriented (SCO) e Asynchronous Connection-Less

(ACL). I collegamenti SCO sono caratterizzati da un invio periodico di pacchetto in singolo slot, principalmente è usato per trasmissione voce che richiede un trasferimento veloce e continuo. Un dispositivo che ha stabilito un collegamento SCO, ha riservati e garantiti certi time slot. I suoi pacchetti sono trattati come pacchetti prioritari e saranno gestiti prima di ogni pacchetto ACL. Un dispositivo con collegamento ACL può inviare pacchetti di lunghezza variabile di 1, 3 o 5 time slot. Ma non ha time slot riservati.

- *Livello Link Manager*: Questo livello implementa il Link Manager Protocol (LMP), che gestisce le proprietà di interfacciamento via radio tra i dispositivi. LMP gestisce l'allocazione di banda per i dati, allocazione di banda e riserva di risorse per traffico audio, autenticazione usando un sistema di negoziazione, pairing tra dispositivi (utilizzo di un codice concordato), cifratura dei dati e controllo della potenza utilizzata. Il controllo della potenza include la negoziazione di modalità in low power activity e determina il livello di potenza del segnale.
- *Livello L2CAP*: Il livello Logical Link Control and Adaptation Protocol (L2CAP) fornisce l'interfaccia tra i protocolli a livello superiore e i protocolli di livello di trasporto. L2CAP supporta il multiplexing di diversi protocolli di alto livello, come RFComm e SDP. Questo permette a una molteplicità di protocolli di condividere l'interfaccia. L2CAP è anche responsabile di segmentazione dei pacchetti e riassemblaggio e per il mantenimento del livello stabilito del servizio.
- *Livello HCI*: L'Host Controller Interface (HCI) definisce un'interfaccia standard per le applicazioni di più alto livello per accedere ai livelli più bassi dello stack. Questo livello non è richiesto dalle specifiche. Il suo scopo è di permettere interoperabilità tra dispositivi e l'uso di protocolli e applicazioni a livello superiore già esistenti.

2.1.2 Comunicazione

Un dispositivo che utilizza Bluetooth è un dispositivo a frequency-hopping spread-spectrum (FHSS) che usa la frequenza libera di 2.4 GHz ISM (Industrial, Scientific, Medical). Nella maggior parte della nazioni ci sono 79 canali liberi, in alcune nazioni se ne possono utilizzare solo 23. L'ampiezza di banda nominale di ciascun canale è di 1MHz.

Le regole FCC 15.247 restringono la massima potenza di picco in uscita a 1 watt e richiedono che almeno 75 di 79 canali siano usati in modalità pseudo casuale. Un dispositivo non può operare su un certo canale per più di 0.4 secondi in un periodo di 30 secondi. Queste limitazioni o restrizioni sono state adottate per minimizzare le interferenze sulla banda ISM, che è anche usata da dispositivi 802.11 b/g, dispositivi HomeRF, telefoni portatili e forni a microonde. Quando un dispositivo Bluetooth si collega ad un altro dispositivo Bluetooth cambia frequenze 1600 volte al secondo per un uso tipico, con una permanenza di 625 μ sec. Quando è nello stato di inquiry o page, cambia le frequenze 3200 volte al secondo con un tempo di permanenza di 312,5 μ sec.

Un dispositivo bluetooth usa tutti i 79 canali e salta in modalità pseudo random con una frequenza di 1600 "balzi" al secondo per una trasmissione standard. Approssimativamente ha un range di 10 metri, sebbene possano essere raggiunti 100 metri con amplificatori. Dal momento che il tranceiver (trasmettitore/ricevitore) è dimensioni ridotte, risulta facile da incorporare sui dispositivi.

Le specifiche bluetooth usano il time division duplexing (TDD) e il time division multiple access (TDMA) per la comunicazione. Un singolo time slot è 625 μ sec di lunghezza, che rappresenta la lunghezza di un singolo pacchetto. A livello Baseband, un pacchetto è composto di un codice d'accesso, un header, e il contenuto vero e proprio.

Figura 2.4 - Formato di un pacchetto

Il codice di accesso contiene l'indirizzo della piconet e di solito è di 72 bit. L'header contiene controlli, codificati con forward error-correcting code (FEC) impostato a 1/3 per alta affidabilità. Questa codifica è una ripetizione di codice, così ogni bit nell'header è trasmesso tre volte. Di solito l'header è di 18 bit, e include l'indirizzo per uno slave attivo. Il contenuto (payload) può variare da 0 a 2745 bit di dati, e può essere protetto da 1/3 FEC (ripetizione semplice, solo per pacchetti SCO), 2/3 FEC (che è in grado di correggere un errore singolo e determinare un duplice errore), 3/3 FEC (che non fa FEC).
Per connessioni SCO, i pacchetti devono essere esattamente lunghi un time slot. Per trasmissioni ACL i pacchetti possono essere 1, 3 o 5 time slot.
La tecnologia Bluetooth usa una trasmissione a pacchetti basata su polling. Tutte le comunicazioni tra dispositivi avvengono tra master e slave, usando il time division duplex (TDD), non c'è comunicazione diretta slave-slave.
Il master interroga ogni slave attivo per determinare se esso ha dati da trasmettere. Lo slave può trasmettere dati solo quando viene interrogato. Deve inviare dati nello slot immediatamente successivo a quello della interrogazione.
Il master trasmette solo nei time slot pari, mentre gli slave trasmettono solo nei time slot dispari. In ciascun time slot, viene usata una diversa frequenza di canale *f* (un passo nella sequenza di "salti").

2.1.3 La Piconet

Le specifiche Bluetooth definiscono una piconet come un rete ad-hoc, che raggruppa dispositivi Bluetooth. In questa configurazione un dispositivo ha il ruolo di master, mentre il resto dei dispostivi è slave. Mentre non ci sono limiti al numero totale di slave in una piconet, in ogni istante ci possono essere al massimo 7 slave. Se ci sono più di 7 slave, i restanti sono inattivi (parked). Il massimo numero di slave "parked" è 255, come definito da SIG, tramite un *parked slave address*. Indirizzando indirettamente i parked slave con le specifiche *Bluetooth device address* è permesso un numero illimitato di slave. Per attivare uno slave parked, il master deve prima mettere nello stato parked uno slave attivo.

Quando due dispostivi Bluetooth entrano nello spazio di comunicazione, provano a comunicare tra loro. Se non c'è una piconet disponibile si precede alla negoziazione. Un dispositivo diventa master (di solito quello che inizia la comunicazione) e l'altro diventa slave.
Ogni dispositivo Bluetooth può funzionare all'interno di una piconet come master, slave o bridge. I ruoli sono temporanei e cessano con l'esistenza della piconet. Il dispositivo master seleziona la frequenza, i tempi (quando ci sono più passi) e l'ordine degli slave. Il master è responsabile di alternare slave inattivi verso altri slave in periodi di inattività.
Master e slave devono scambiarsi indirizzo e clock per permettere allo slave di unirsi alla piconet. I dispositivi Bluetooth hanno un identificativo unico (Global ID) che viene usato per creare uno schema a più passi. Il master condivide il suo Global ID e clock con ciascuno slave della sua piconet. Lo slave si deve sincronizzare con il master.
Un bridge o gateway Bluetooth collega due o più piconet per una comunicazione a più passi. Il bridge comunica con tutte le piconet a lui connesse sincronizzando il proprio clock con ciascuna piconet, tuttavia può comunicare con solo una piconet alla volta. Il bridge introduce overhead spostandosi da un clock di una piconet al clock di un'altra piconet e può diventare un collo di bottiglia. Un bridge può essere slave in tutte le piconet a cui è connesso, o può essere master in una e slave nelle altre. L'interconnessione di due o più piconet via bridge determina la formazione di una Bluetooth scatternet.

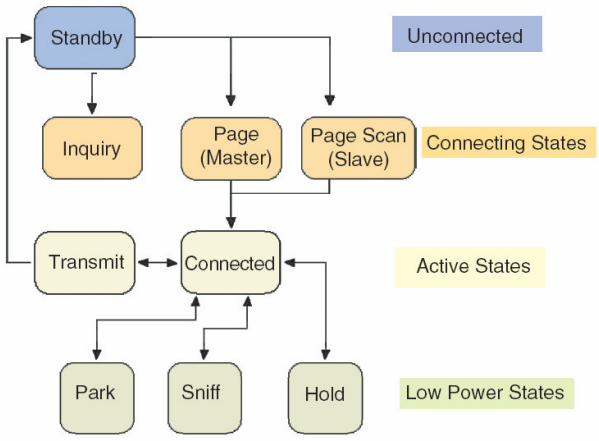


Figura 2.5 - Stati di un dispositivo Bluetooth

Un dispositivo bluetooth può essere nei seguenti stati: standby, inquiry page, connected , transmit, hold, park o sniff, come in figura 2.5.
Un dispositivo è in modalità *Standby* quando è acceso, ma non è collegato a una piconet. Va nello stato *Inquiry* quando invia richieste per trovare altri dispositivi a cui potersi collegare. Un master di una piconet esistente può essere nello stato di *Page*, che invia messaggi alla ricerca di dispositivi che può associare alla sua piconet.
Quando la comunicazione tra il master e il nuovo dispositivo ha successo, il nuovo dispositivo assume il ruolo di slave, entra nello stato *Connected* e riceve un indirizzo attivo. Lo slave mentre è connesso, può trasmettere dati quando il master lo interroga. Durante la trasmissione dei dati lo slave è nello stato *Transmit*. Alla fine della trasmissione ritorna nello stato *Connected*.
Lo stato *Sniff* è uno stato di basso consumo in cui lo slave "riposa" per un predeterminato lasso di tempo.
Lo stato *Hold* è un altro stato di basso consumo in cui lo slave non è attivo per un certo intervallo di tempo. Non c'è trasmissione dati durante lo stato di hold.
Quando un dispositivo slave non ha dati da trasmettere, il master può decidere se metterlo nello stato *Parked*. Lo slave rilascia il suo indirizzo attivo nella piconet. L'indirizzo sarà dato a un altro slave nella piconet che il master ha riattivato dallo stato parked.

2.2 Supporto per l'Handoff su reti mobili IP Bluetooth

BLUEtooth Public ACcess (BLUEPAC) [ALM99] è un'architettura di rete sviluppata da una sinergia dell'ambiente accademico con l'industria per permettere a dispositivi Bluetooth l'accesso alla rete pubblica (es. in stazioni e aeroporti).
In un'area BLUEPAC, il dispositivo bluetooth dovrebbe essere in grado di muoversi tra diversi access point mantenendo lo stesso indirizzo IP.
Sono stati proposti protocolli [BAS00] per il supporto alla mobilità al livello 3 OSI e livello 2 per supportare la mobilità e l'handoff tra differenti access point. Sono stati necessari adattamenti per permettere ai datagrammi IP di passare da dispositivi Bluetooth agli access point.

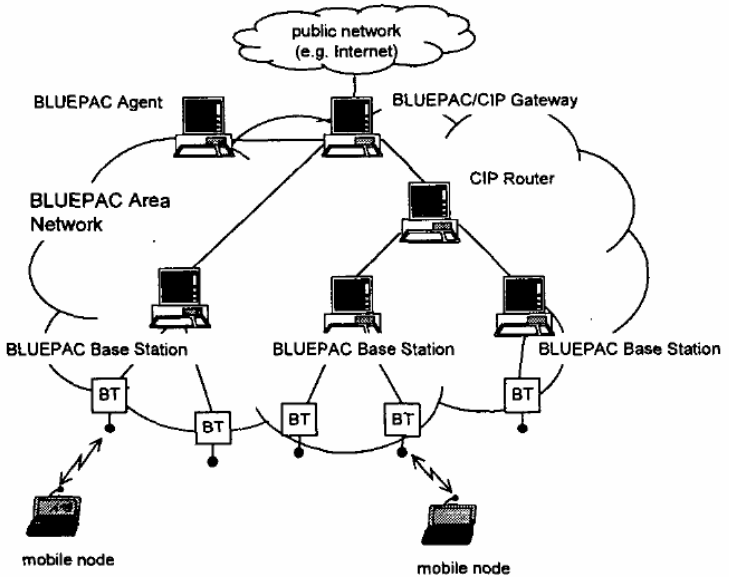


Figura 2.6 - Elementi funzionali di una rete di riferimento BLUEPAC

La *BLUEPAC area network* è la parte fondamentale della rete, a cui accedono i dispositivi Bluetooth (in figura *mobile node*).
Nella maggior parte dei casi la rete BLUEPAC è una rete cablata. Le *BLUEPAC Base Station* sono access point. I dispositivi mobili Bluetooth usano gli access point per accedere alla rete. Così una base station è composta da un'interfaccia verso la rete fissa e almeno un Bluetooth tranceiver (BT). Il BLUEPAC Gateway può connettere la rete BLUEPAC con le rete pubblica (es. Internet). Il *BLUEPAC Agent* coordina le comunicazioni con i nodi mobili che si presentano. Questo include azioni come l'assegnamento di un indirizzo IP. I *Cellular IP (CIP) Router* giocano un ruolo importante nella rete BLUEPAC, e gestiscono il routing fra nodi mobili.

2.2.1 Cellular IP in BLUEPAC

Il protocollo Cellular IP (CIP) è stato usato per Bluetooth Public Access (BLUEPAC) per supportare l'handoff. La rete CIP per BLUEPAC è composta da gateway che sono connessi a internet o altre reti locali e router che raggiungono sia i gateway che le BT. CIP fornisce aggiornamenti veloci delle tabelle di routing e permette variazioni frequenti. I dispositivi mobili inviano pacchetti ad internet tramite il gateway.

2.2.1.1 Limitazioni di CIP in BLUEPAC

Il protocollo CIP ha due limitazioni:

- L'handoff provoca interruzioni: si verificano perdita di pacchetti e ritardi prima della caduta della connessione. La caduta di connessione viene determinata quando scatta un timeout.
- Ricerca del nuovo access point: dopo la perdita di connessione, è il dispositivo mobile che si deve preoccupare di trovare un nuovo access point, ma non ha conoscenza degli indirizzi degli access point e deve entrare nello stato di inquiry per ottenerli. Questo porta ad un handoff lento. In aggiunta periodicamente gli access point devono entrare nello stato di inquiry e page, per trovare nuovi dispositivi, questo peggiora le risorse di banda.

Un altro svantaggio è che il CIP gira sul dispositivo mobile che abbia il supporto IP. Se i dispositivi portatili non lo implementano il CIP non può essere usato.

2.2.2 Neighborhood Handoff Protocol

E' stato proposto un altro protocollo di handoff, che prende il nome di Neighborhood Handoff Protocol (NHP), per permettere un handoff veloce [KAA02].

Questo metodo previene la perdita di pacchetti durante l'handoff e cerca di ottenere un uso efficiente della banda. Questo protocollo non richiede nessun cambiamento ai nodi mobili e deve essere implementato sull'access point sopra il livello Bluetooth standard. NHP elimina il tempo di attesa della procedura di inquiry dell'handoff e porta ad un rilevamento rapido della perdita di connessione. Gli access point vicini alle entrate sono candidati ad essere *entry point*. Sono implementati sullo stesso hardware degli access point, ma svolgono una funzione diversa. I dispositivi mobili possono accedere alle rete solo tramite gli entry point e non da ogni arbitraria punto di accesso.

Le operazioni del protocollo consistono di due attività:

- *Entry*: Un entry point costantemente esegue inquiry. Ogni volta che rileva un nuovo dispositivo stabilisce una connessione ottiene un indirizzo e informazioni sul clock. L'entry point chiude la

connessione e passa le informazioni di clock e indirizzo ricevute dal nuovo dispositivo al più vicino access point. L'access point e l'entry point sono collegati tramite LAN. Quell'access point temporaneamente sospende le sue comunicazioni e va nello stato di page. Dal momento che le informazioni di indirizzo e clock sono già note, la procedura si conclude in meno di 1.28 secondi.

- *Handoff*: Lo schema di polling usato è il round robin, in cui il master interroga ciascuno slave uno dopo l'altro, con la possibilità di variare la durata dei pacchetti. Lo slave deve sempre confermare la ricezione di pacchetti dal master, anche se non sono stati inviati dati. Se lo slave non risponde, viene rilevata la perdita di connessione (al max in 50 ms). Non ci sono pacchetti persi dal momento che al massimo un pacchetto può essere inviato durante un ciclo di polling, e questo sarà ritrasmesso dal livello Bluetooth.

Una volta che è rilevata la perdita di connessione, ne deve essere stabilita una nuova. Definiamo *neighborhood set* di un access point A, l'insieme di tutti gli access point nel cui raggio di copertura può trovarsi un dispositivo mobile che era raggiunto dall'access point A, fino a 50 ms prima. Appena è rilevata la perdita di connessione l'access point corrente invia il clock e indirizzo del dispositivo perso a tutti i suoi access point neighborhood. Sono inoltrati anche tutti i pacchetti non inviati. Ciascun access point va nello stato di page usando il clock e l'indirizzo ricevuto dal vecchio access point.

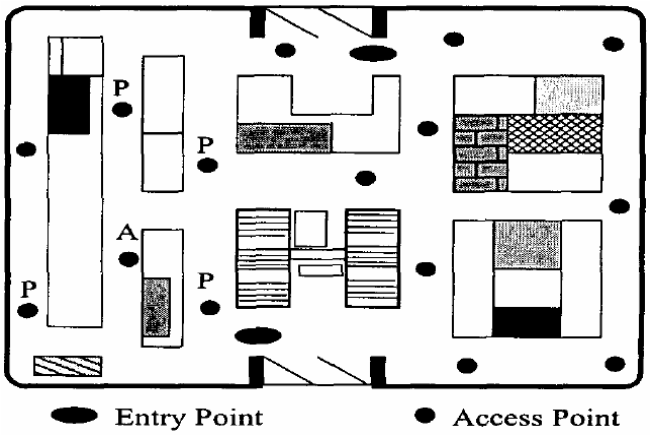


Figura 2.7 - Disposizione di access point ed entry point in una rete Bluetooth con NHP

Non viene ritentata la procedura di paging, assumendo che se il dispositivo non è stato trovato, sarà stato trovato da un altro access point o si è spostato fuori dal range della rete. La connessione può essere ripristinata in 1.28 secondi.
In figura 2.7 vediamo ad esempio la pianta di una sala. Consideriamo l'access point A. E' facile vedere che un nodo che era nel range di A fino a 50 ms prima, si è potuto muovere solo verso gli access point indicati con P.
E' possibile ridurre ulteriormente il tempo di handoff e migliorare la banda utilizzando complessi meccanismi di predizione degli spostamenti, sia su dati topologici che di apprendimento.

2.3 Quality of Service in reti IP Bluetooth Ad-Hoc

Un dispositivo Bluetooth che accede a servizi multimediali su internet, può farlo attraverso una piconet o una scatternet.
Quando un BT vuole ricevere un video memorizzato su un server video remoto, il BT (potrebbe essere slave in una picocell) inizierà la procedura di connessione con la picocell master. Come descritto in [CHW03] il master inizia la procedura di connessione con il video server attraversando il sistema Bluetooth-IP. Durante la connessione, il video stream passerà attraverso la rete fino al master e al BT.

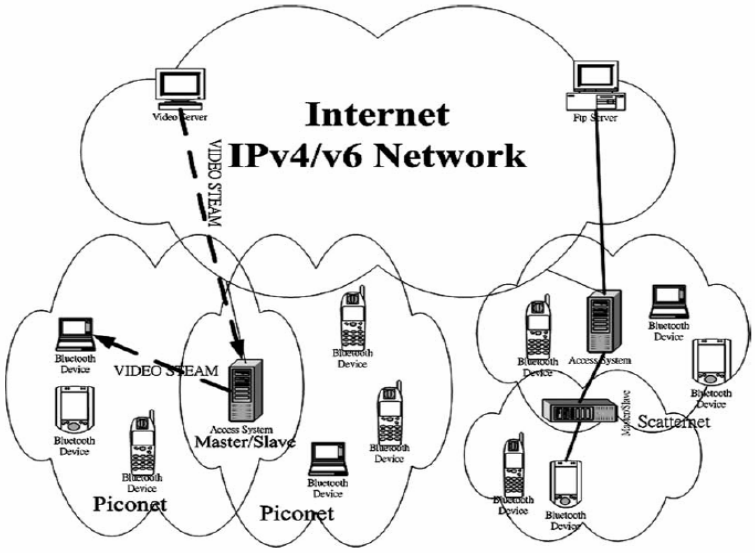


Figura 2.8 - Ambiente di rete Bluetooth-IP

La Qualità of Service (QoS) è critica durante la trasmissione su diversi segmenti di rete. Per fornire servizi servizi IP su bluetooth con diverse livelli di qualità, è stato proposto un sistema che indicheremo con QoS-centric cascading mechanism che ha certe garanzie di servizio. I vari blocchi si possono vedere in figura 2.9.

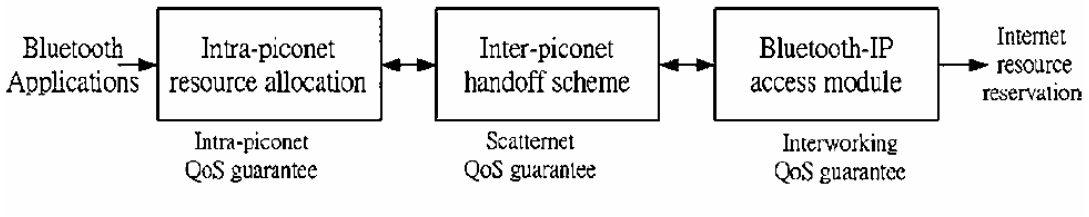


Figura 2.9 - Blocchi del QoS-centric cascading

Il funzionamento si basa su tre moduli: intra-piconet resource allocation, inter-piconet handoff e Bluetooth-IP access system. Questi moduli sono basati sul protocollo Bluetooth Network Encapsulation Protocol (BNEP).

2.3.1 Allocazione di risorse intra-piconet

Ci sono due tipi di servizi definiti per l'ambiente Bluetooth: Synchronous Connection Oriented (SCO) e Asynchronous Connectionless Link (ACL). L'ACL può essere configurato per fornire la QoS richiesta. Possono essere impostate i timeout per i dati prevenendo inutili ritrasmissioni. La conferma di ricezione può essere ottenuta entro 1.25 ms. Questo ritardo è abbastanza piccolo per permettere ritrasmissioni per applicazioni real-time. L'ACL supporta anche ampiezza di banda asimmetrica e variabile.

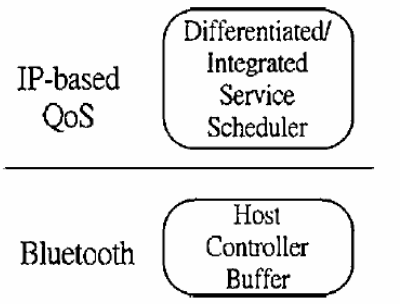


Figura 2.10 - Struttura Bluetooth IP QoS

Attualmente, l'architettura per QoS su IP è basata soltanto a livello IP, facendo buffering dei pacchetti e scheduling a livello di collegamento con l'access point. Come mostrato in figura 2.10, l'architettura per QoS a livello di rete fornisce diversi servizi a livello applicativo.
Questi servizi ad alto livello sono abbastanza dipendenti dal particolare scenario.

2.3.2 Handoff inter-piconet

L'ambiente inter-piconet è soggetto a molti fattori di influenza. Primo la formazione di reti Bluetooth è spontanea e il problema di creazione di scatternet non è stata ben definita nelle specifiche Bluetooth.
Sono stati sviluppati protocolli efficienti per routing, ma in generale inter-piconet sono colli di bottiglia per la scatternet. Come abbiamo visto la connessione richiede due passi: inquiry e paging. Due situazioni sono state discusse in BLUEPAC.
Quando una access point è master, il BT si unisce alla piconet come slave. L'access point può efficientemente controllare il traffico verso Internet. Lo svantaggio di questa scelta è che l'access point deve periodicamente entrare nello stato di inquiry per rilevare nuovi dispositivi. Questo interrompe il trasferimento di pacchetti verso Internet.
Nella situazione che il BT è master, l'access point risulta in più piconet. Tuttavia il traffico risulta più complesso quando l'access point deve commutare tra diverse piconet.
Lo scheduler non adatto per handoff senza interruzioni in applicazioni real-time. Per risolvere questo problema è stato proposto il Next Hop Handoff Protocol (NHHP) per un veloce handoff.

2.3.3 Sistema di accesso Bluetooth-IP

La differenza tra Bluetooth piconet e IP LAN è lo stack di protocolli come in figura 2.11.
Queste differenze sono sui livelli più bassi dei sette livelli OSI. Il livello più basso è responsabile del collegamento e indirizzamento. Le funzioni di accesso permettono connessioni senza particolari conoscenze o interazioni.

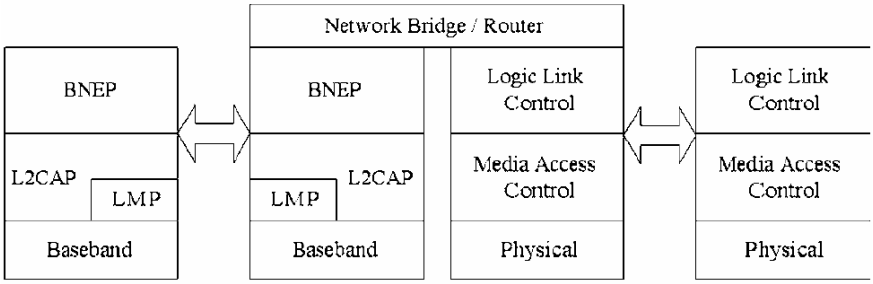


Figura 2.11 - Stack di protocolli per Bluetooth IP interworking

Il sistema di accesso Bluetooth-IP gioca un ruolo di bridging/routing per il traffico tra LAN e piconet.

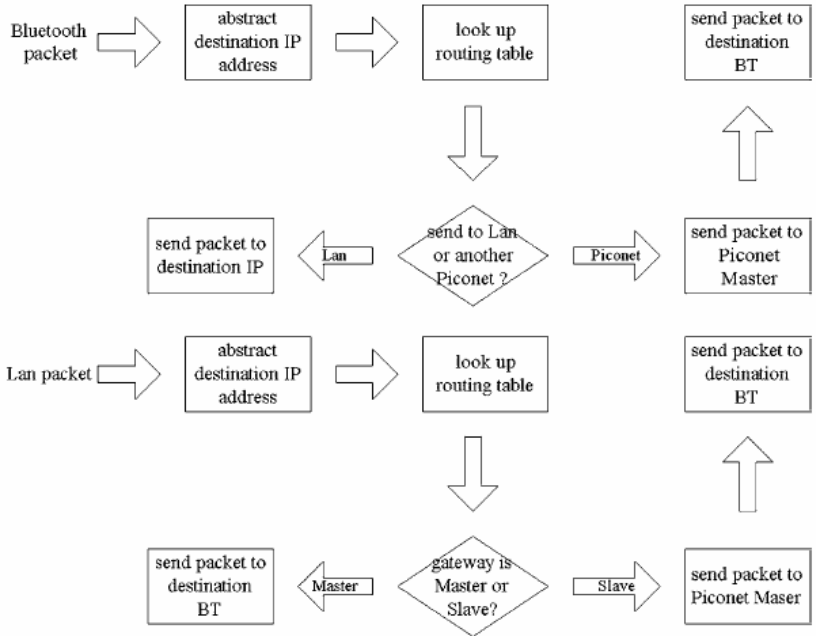


Figura 2.12 - Scenario operativo di Bluetooth IP interworking.

Per permettere l'interconnessione, la piconet deve comunicare con la LAN. Due diversi protocolli devono essere combinati tra loro per formare un nuovo protocollo di comunicazione come riportato in figura 2.12. Usando le specifiche dello stack di protocolli, lo scenario di indirizzamento è così identificato:

- Ciascuna host di una LAN ha assegnato un indirizzo IP. Ciascun host così possiede due indirizzi: un indirizzo IP e un MAC address.
- Ciascuna piconet BT acquisisce due indirizzi, un BT address (BD_ADDR) e un IP address.

- Deve essere costruita una tabella di routing per l'interconnessione. E' necessario fare il lookup del destinatario per conoscere il BD_ADDR o MAC a cui inviare i pacchetti.

2.4 Streaming in Bluetooth

Le reti IEEE 802.11 e Bluetooth sono viste come estensioni della rete Internet, e diventa un'esigenza il supporto su queste reti a uno streaming di contenuti multimediali efficiente. In riferimento alla tecnologia Bluetooth sono state proposte soluzioni che implementano un'architettura in grado di fornire il supporto per il protocollo di streaming audio sfruttando lo standard Audio/Video Distribution Transport Protocol (AVDTP) specificato per Bluetooth.
I profili Bluetooth sono molto importanti e possono essere impiegati per l'interoperabilità di applicazioni che implementano lo stesso profilo su diverso hardware. I profili che sono in grado di supportare audio, anche in streaming sono gli Advanced Audio Distribution Profile (A2DP), Local Area Network (LAN) e Bluetooth Network Encapsulation Specification (BNEP).
L' Audio Video Working Group [AVWG05] definisce un profilo con un insieme di procedure per stream sincrono di audio di alta qualità sul protocollo Logical Link Control and Adaption Protocol (L2CAP) verso un altro dispositivo [PRA03].

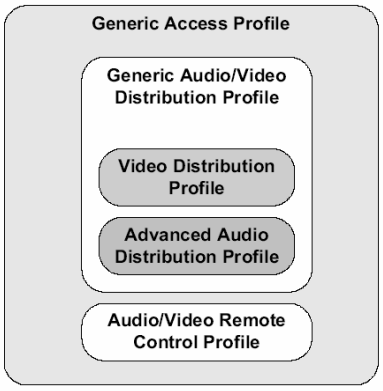


Figura 2.13 - L'A2DP nel contesto di altri profili.

Sul protocollo AVDTP sono state aggiunte funzionalità di riproduzione audio tra dispositivi Bluetooth rivolte alla Compressione/Decompressione (CODEC) con supporto per un alto impiego di banda tra dispositivi

Bluetooth e host Internet (sfruttando la connettività BNEP per IP) [ZES04], come si vede in figura 2.14.

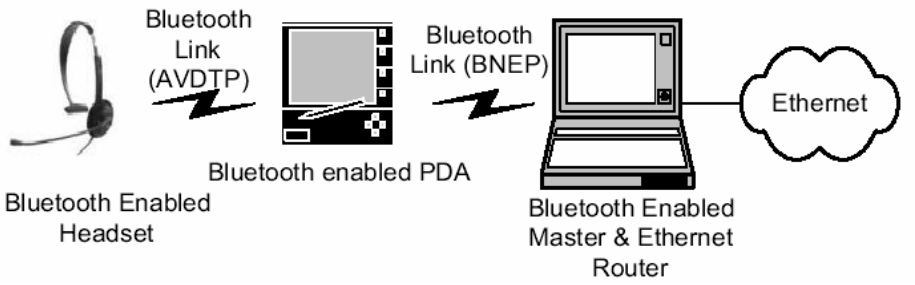


Figura 2.14 - L'audio streaming (via AVDTP) è realizzato tra PDA con Bluetooth e auricolari Bluetooth.

Benché audio o voce possano essere mandate in stream a un dispositivo di solito è il dispositivo che fa streaming che poi lo riproduce. Nella figura 2.15 si riporta invece un'estensione a questo schema in cui lo stream può provenire da un host di rete e il dispositivo ricevente può spostarsi all'interno della rete Bluetooth. L'estensione sfrutta il protocollo AVDTP su reti IP con il protocollo BNEP.

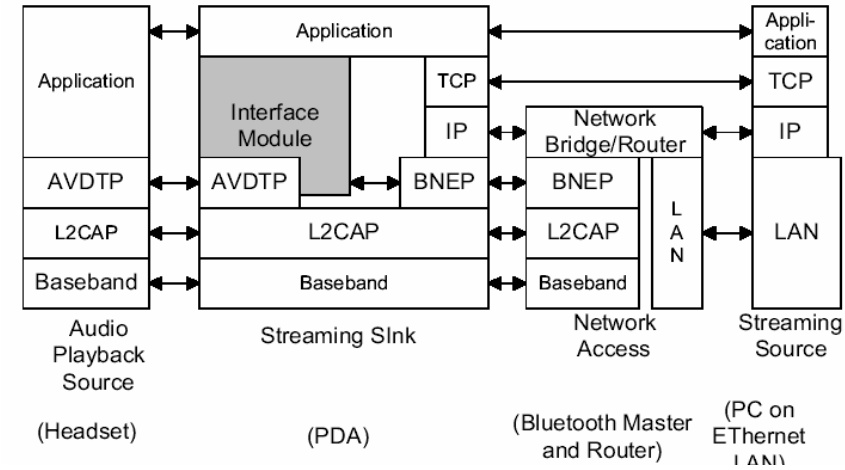


Figura 2.15 - Architettura per lo streaming e riproduzione audio per auricolari Bluetooth.

2.4.1 Middleware con ubiQoS per streaming audio

Altre ricerche per mantenere un certo livello di qualità di servizio sono state proposte da [BEP04, BES04]. E' stato sviluppato uno strato software che prende il nome di ubiQoS, che dà garanzie di QoS per audio streaming

su dispositivi Bluetooth utilizzando un proxy per adeguare la qualità del servizio verso i client, utilizzando diversi tipi di collegamenti Bluetooth.
I proxy lavorano sulla rete cablata e possono spostarsi, per seguire il movimento del dispositivo mobile secondo le necessità.
La modalità di collegamento ACL fornisce un servizio orientato al pacchetto, mentre il collegamento SCO è orientato alla connessione e progettato per applicazioni con vincoli temporali come streaming audio. Questi due tipi di collegamenti hanno diverse QoS. L'utilizzo di un collegamento SCO può lasciare poca banda alla piconet da utilizzare per un collegamento ACL. In funzione del servizio richiesto, il servizio di streaming su Bluetooth dovrebbe essere in grado di scegliere il tipo di collegamento più adatto.
E' stato proposto a livello middleware la soluzione ubiQoS, per gestire la QoS in un ambiente best-effort wireless Internet.
I proxy si trovano sulla rete cablata e si affacciano sulla rete wireless. Nella piconet il proxy lavora come master e aggiusta dinamicamente il livello di QoS di stream audio. I proxy sono implementati come agenti mobili, cioè entità attive che possono spostarsi da un nodo ad una altro durante la loro esecuzione, portandosi dietro il codice e lo stato raggiunto.
Per permettere la portabilità è stata realizzata [BES04] un'interfaccia utilizzando Java chiamata JSR82+SCO che estende le Java API Bluetooth (JSR82) con il supporto di collegamento SCO. Da un lato è stata sviluppata un'implementazione che estende le JavaBluetooth per supporto a ACL JSR82, dall'altro è stata sviluppata modificando il supporto SCO Java, integrandolo con la Java Native Interface.
ubiQoS associa ad ogni client Bluetooth un'entità, chiamata *shadow proxy*, e applicazioni chiamate *QoS adapter*. Shadow proxy e QoS adapter sono ospitati nei *places*, ambienti di esecuzione che offrono servizi base per la comunicazione tra agenti mobili e migrazione. I place di solito rappresentano nodi della rete e possono essere raggruppati in domini a seconda della località.

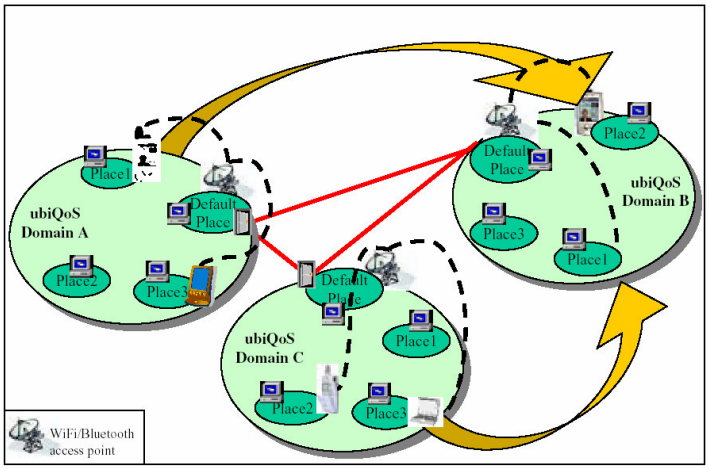


Figura 2.16 - Spostamento del dispositivo mobile su domini ubiQoS

La figura 2.17 concentra l'attenzione su un singolo dominio. Alla richiesta del client di flusso audio lo shadow proxy interroga il servizio dei nomi ubiQoS per ottenere la lista di tutti i server audio. Il proxy poi recupera i profili applicabili ad un certo utente (caratteristiche del dispositivo e preferenze) e il flusso audio. In base al profilo il proxy decide come adattare la trasformazione con i QoS adapter.

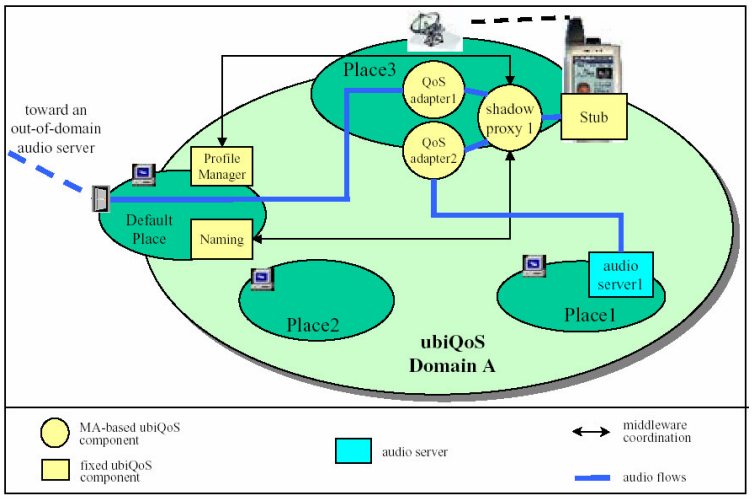


Figura 2.17 - Il middleware ubiQoS mentre distribuisce il flusso audio tra client wireless

I QoS adapter ricevono il flusso audio , operano le trasformazioni decise dal proxy a loro associato e mandano il risultato a un audio player. L'implementazione corrente di QoS adapter è basata su JMF. Per il

trasporto e controllo di il QoS adapter usa le API JMF utilizzando il Real-time Transport Protocol (RTP) e il corrispondente RTCP control protocol. Shadow proxy e QoS adapters in generale sono portabili su ogni piattaforma che abbia installata la Java VM.

2.5 Applicazioni Video Streaming su dispositivi mobili

Nel contesto di applicazioni multimediali ci sono molte ricerche per migliorare la qualità video su dispositivi mobili.
L'adattamento può essere realizzato in molti modi e per scopi diversi. Di solito il processo di adattamento viene impiegato per ridurre l'occupazione di banda. Le tecniche di adattamento rappresentano una soluzione che permette buona flessibilità, anche se spesso comportano il degrado dell'accuratezza del media.
In soluzioni client-server il processo di adattamento viene eseguito lato server in modo da ridurre il consumo delle risorse e delle batterie del client. In soluzioni peer-to-peer il processo di adattamento viene realizzato da chi trasmette, molto spesso utilizzando implementazioni hardware più efficienti rispetto ad applicativi che comportano i problemi di eccessivo uso di risorse e batterie come prima accennato. E' possibile anche pensare ad uno scenario in cui tra i pari ci sia una (o più) entità proxy, che permetta una adattamento.
La qualità del media può venire concordata all'inizio, o modificata nel corso della comunicazione con un processo di adattamento dinamico a seconda della qualità del collegamento.

2.5.1 Adattamento localizzato

Molte tecniche di adattamento lavorano sull'intero frame, con il risultato che tutto il video risulta adattato. Una soluzione diversa e mirata parte dal presupposto che ci possono essere all'interno della scena delle parti che hanno una rilevanza maggiore e che richiedono un'attenzione maggiore rispetto ad altre che in caso di necessità possono venire trascurate [COD05]. L'idea di base è di ridurre le dimensioni del media rispetto all'originale permettendo all'utente di scegliere la regione di più interesse. Questa strategia per applicazioni real-time [COD05] adotta una nuova tecnica di riduzione spaziale.

Durante la fruizione del servizio l'utente può cioè scegliere dinamicamente la regione di interesse. La forma di questa regione è rettangolare e può essere selezionata arbitrariamente all'interno del frame. All'inizio della fornitura del servizio multimediale il video è trasmesso senza riduzione spaziale. Quando il sistema di rilevamento di qualità nota una degradazione della qualità, fa partire il processo di adattamento. L'adattamento consiste nella riduzione dell'area di interesse, in accordo con la forma indicata dall'utente.
Nel caso peggiore, quando c'è maggior degrado nella trasmissione, verrà trasmessa solo la regione di interesse. Il processo inverso, cioè l'espansione della dimensione del video viene attuata in modo simile quando le condizioni di trasmissione sono sufficienti a trasmettere una regione più ampia.

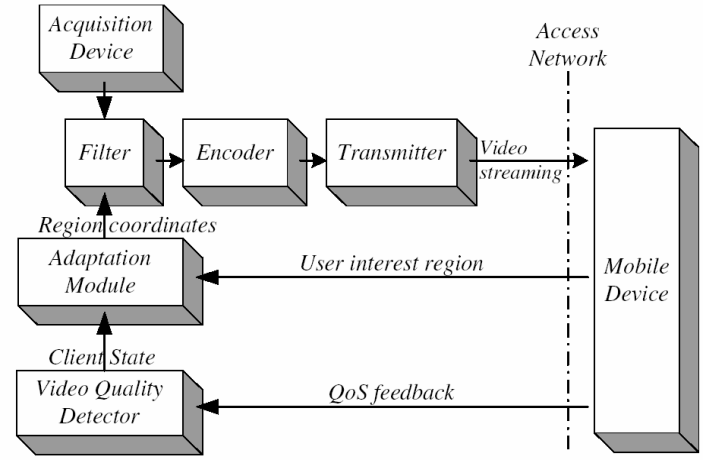


Figura 2.18 - Modello del sistema di adattamento localizzato

L'architettura di questo sistema di adattamento adotta il paradigma client-server, in cui il processo di adattamento viene realizzato lato server. Questo permette la riduzione di consumo di risorse sul dispositivo mobile. Come da figura 2.18 il blocco Video Qualità Detector fornisce un indice della qualità del servizio. Il modulo di adattamento utilizza questa informazione per scegliere se ridurre o allargare la regione di interesse selezionata dall'utente.
Quando l'indice si abbassa sotto una soglia predefinita, il modulo di adattamento attiva il processo di riduzione della regione. Quando l'indice

aumenta oltre a una certa soglia, che vuol dire che la qualità video è eccellente, la regione viene allargata di una certa percentuale. Quando il modulo di adattamento rileva un cambiamento nello stato del client, o l'utente seleziona una nuova regione, calcola la nuova regione e trasmette questa informazione al modulo Filter. L'operazione di filtraggio è applicata ad immagini video non compresse che vengono poi compresse con un opportuno encoder e trasmesse.

2.6 Conclusione

In questo capitolo abbiamo parlato di tecnologia Bluetooth, delle topologie di rete di accesso e della possibilità di collegare dispositivi Bluetooth a reti LAN o Internet. Abbiamo visto gli accorgimenti che possono essere presi durante il collegamento agli access point da un dispositivo che si sposta. Abbiamo parlato di qualità del servizio (QoS) riferita anche all'idea di un servizio continuo, come può essere quello di streaming multimediale, sempre con l'idea di un collegamento che ha un inizio, uno svolgimento ed una fine (sessione). Sono state proposte alcune soluzioni adottate nello streaming su dispositivi bluetooth di contenuti audio e video, contemplando l'architettura di protocolli utilizzati.

CAPITOLO 3

Analisi del Sistema

In questo capitolo analizzeremo le motivazioni che porteranno alla realizzazione del nostro progetto. Partendo dalla descrizione degli streaming tradizionali motiveremo le scelte che prenderemo nella realizzazione dell'architettura. Lavoreremo sempre con l'idea di sviluppare un'applicazione per flussi audio, con collegamento wireless Bluetooth in un contesto che può essere LAN o ampio come Internet.
Il dispositivo mobile (client) richiederà il servizio ad un server. In questo capitolo analizzeremo l'architettura del sistema scomponendola in queste sue due parti descrivendo le modalità di interazione.

3.1 Scopo del progetto

Lo scopo di questa tesi è di fornire una soluzione alternativa allo stream audio tradizionale per la riproduzione continua del media su collegamento Bluetooth.
Durante la trasmissione di un flusso multimediale la qualità della comunicazione può variare in un ambiente best-effort come Internet e l'impiego del collegamento wireless Bluetooth necessita di una serie di considerazioni aggiuntive.
Il dispositivo mobile utilizza un collegamento che presenta caratteristiche di comunicazione soggette a molteplici fattori, come interferenze con altri apparati nelle vicinanze o dipendenti da movimento. In un collegamento wireless l'accesso viene fornito da un access-point connesso alla rete fissa. Durante lo spostamento può essere opportuno passare da un access-point ad un altro, o perché presenta un collegamento migliore o perché l'altro non risulta più raggiungibile. Questo passaggio prende il nome di *handoff* e deve risultare il più indolore possibile. In ogni caso c'è un'interruzione della comunicazione e del flusso dati. Vogliamo che a seguito di handoff, o di perdita di collegamento, la riproduzione audio sul nostro dispositivo non ne risenta.
Il dispositivo deve permettere di continuare ad eseguire nell'attesa che una nuova connessione si renda disponibile.

Gli stream vengono utilizzati perché permettono di iniziare la riproduzione del contenuto multimediale dopo un periodo di buffering senza dovere attendere di avere scaricato l'intera presentazione, quindi comportano un risparmio di tempo e memoria. Alcune applicazioni non troverebbero altre soluzioni, basti pensare ad uno streaming di una radio in Internet. Nei dispositivi mobili poi la limitazione di memoria non permetterebbe di scaricarsi presentazioni oltre una certa durata. Scaricare e riprodurre significa focalizzare l'attenzione su una certa porzione temporale del flusso più che sulla durata del media stesso. In un flusso live poi non ha senso parlare di durata dal momento che continua indefinitamente.

E' necessario che il flusso in streaming arrivi con una certa continuità per garantire a sua volta una continuità di riproduzione, nel caso contrario la riproduzione verrà interrotta per mancanza di dati, e solo dopo una successiva fase di buffering si potrà riprendere la presentazione.

Nelle reti cablate, a meno di utilizzare protocolli che garantiscano una comunicazione affidabile con una certa qualità di servizio, che non sempre l'infrastruttura supporta, non c'è nessuna garanzia di consegna entro un tempo prefissato.

Nei collegamenti wireless si aggiungono altri problemi che condizionano la qualità del servizio. Sono sempre possibili interferenze sulla tratta radio dovute ad altri dispositivi o condizioni ambientali. Utilizzando la tecnologia bluetooth lavoriamo sulla banda di 2.4 GHz ISM (Industrial, Scientific, Medical) che in qualità di banda libera viene utilizzata anche da altri dispositivi, come ad esempio collegamenti wireless 802.11, forni a microonde e apparecchiature mediche.

Poniamo la nostra attenzione sulla qualità del servizio utilizzando un'infrastruttura che non dà garanzie in questo senso.

I flussi tradizionali sono in grado di adattare le caratteristiche del media che trasmesso in funzione delle condizioni di rete rilevate diminuendo la qualità in caso di congestione o aumentandola nel caso opposto. Interruzioni sulla tratta radio possono portare ad una interruzione del servizio.

Lo scopo di questo progetto è di continuare a fornire il servizio a fronte di interruzioni di entità limitata. Per interruzioni maggiori si decide l'azione

da intraprendere quando la comunicazione torna disponibile. Per fare questo occorre mantenere una sessione si lavoro che stabilisca da dove ripartire.

3.2 Descrizione del progetto

Il dispositivo mobile si trova in un contesto che potrebbe essere una rete locale con punti di accesso Bluetooth, o un contesto più grande, come Internet.

Può in ogni momento richiedere l'erogazione del servizio al server. Il server può ricevere richieste da diversi client. Il primo client farà partire il processo di elaborazione del flusso audio, i successivi riceveranno informazioni per unirsi al download della presentazione. Notiamo che non si tratta comunque di trasmissione multicast.

Il dispositivo mobile nella richiesta specifica il formato audio desiderato, la richiesta può essere soddisfatta se è il primo client che esegue la richiesta e quindi l'elaborazione non è ancora partita, altrimenti utilizzerà le impostazioni dettate dal primo client.

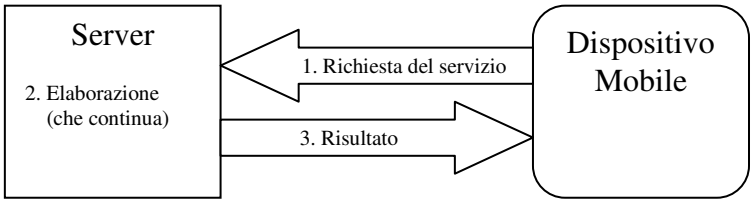


Figura 3.1 - Schema generale di funzionamento

Da notare, come mostrato in figura 3.1 che mentre la richiesta al passo 1 viene eseguita solo all'inizio del protocollo di comunicazione , la fase 2, di elaborazione e 3, di download del processo di elaborazione continuano per tutta la durata della fornitura del servizio.

Lato client si inizia a scaricare e ad eseguire acquisendo con un certo anticipo le risorse da riprodurre per superare eventuali cadute di connettività. Il client per tutta la durata di funzionamento manterrà la sessione e a fronte di eventi come interruzioni prolungate agirà di conseguenza.

Al ripristino del collegamento deciderà, a secondo del contenuto che stava ricevendo, se ha senso ripartire da dove di era interrotto (es. brano musicale) o da un altro punto (es. cronaca di una partita).
Per distribuire il carico di lavoro e rendere un servizio più vasto è possibile pensare anche all'impiego di proxy, che fanno cache delle presentazioni memorizzate sul server per servire una molteplicità di client senza sovraccaricare il server che sta eseguendo l'elaborazione del flusso.

3.3 Architettura del progetto

Andiamo ora a vedere lo schema a blocchi del sistema. La realizzazione del progetto si basa su una architettura che prevede due entità principali: client (il dispositivo mobile) e server.
In figura 3.2 troviamo una schematizzazione delle entità in gioco con le unità funzionali.

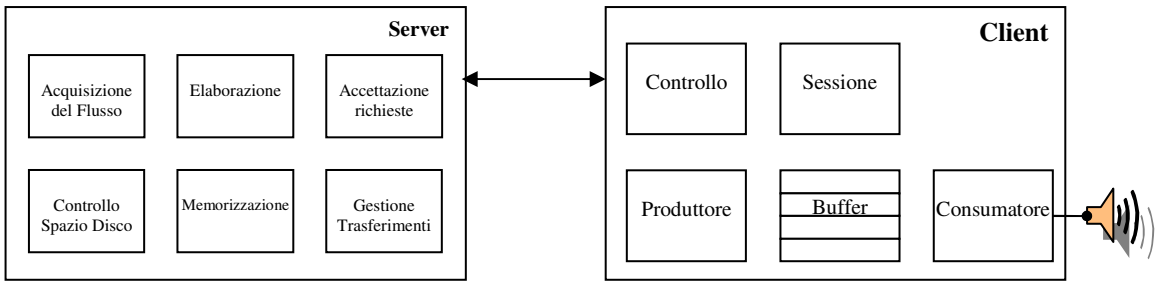


Figura 3.2 - Architettura del progetto

Nei prossimi paragrafi vedremo nel dettaglio il funzionamento di questi componenti.

3.3.1 Lato Server

Il server web è caratterizzato dai blocchi funzionali riportati in figura 3.3.
Vediamoli nel dettaglio:

- *Acquisizione del flusso*: si occupa di interfacciarsi con la periferica di acquisizione che fornisce lo streaming audio.
- *Elaborazione*: è il blocco fondamentale del sistema. Quando riceve l'ordine di acquisire il flusso, inizia ad elaborare il media sulla base delle impostazioni sul formato audio ricevuto e lo passa al modulo di Memorizzazione.

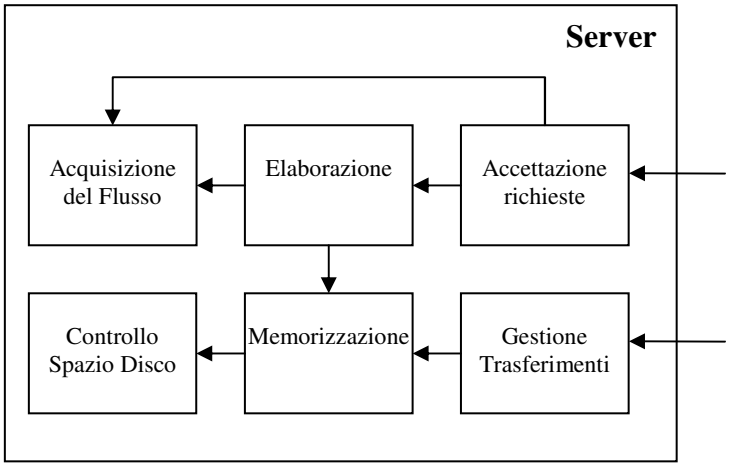


Figura 3.3 - Struttura del server

- *Accettazione richieste*: è in ascolto, in attesa di richieste. Mantiene lo stato dell'elaborazione se già partita, altrimenti un client che si rivolge a lui determina l'avvio del processo. Dopo la prima richiesta, le successive non fanno partire una nuova elaborazione. Il client viene informato che il flusso è già disponibile per essere scaricato specificandone il percorso.
- *Memorizzazione*: riceve il risultato dell'elaborazione e memorizza il media su file. La durata di ciascuna clip è concordata in fase di accettazione. Le clip sono dei file audio di qualche secondo ordinate con un numero progressivo
- *Controllo spazio disco*: nel corso dell'acquisizione questo modulo controlla lo spazio su disco impiegato dalle presentazioni onde evitare che si esaurisca. A seconda delle caratteristiche del sistema si decide dopo quanto tempo eliminare una clip. Questa scelta è molto importante e determina quanti minuti o ore di presentazione acquisita sono disponibili. Un client potrebbe decidere ad esempio di ascoltare la presentazione in differita di qualche ora. Lo spazio disco viene quindi controllato e quando si ritiene opportuno le clip più vecchie vengono cancellata. Un sistema più semplice per la gestione del disco sarebbe quella di ricominciare la numerazione dopo un certo tempo. In questo caso però potrebbero nascere

problemi se un client richiede una clip con un ritardo superiore al tempo di aggiornamento, perché al posto di quella richiesta ne trova una che l'ha sovrascritta.

- *Controllo trasferimenti*: questo modulo si occupa di rendere disponibile la presentazione a chi ne faccia richiesta.

Sia il modulo di accettazione richieste che quello di controllo dei trasferimenti comunicano con il client tramite il protocollo HTTP. La scelta è stata fatta per il semplice motivo che come protocollo è implementato nella maggior parte dei dispositivi e lavorando ad alto livello evitiamo dettagli che possono dipendere da particolari dispositivi.

3.3.2 Proxy

Per alleggerire il carico di lavoro del server si può pensare di inserire tra il server e i client uno o più proxy, la fase iniziale avverrà sempre tra il client e il server, ma poi il client si rivolgerà al proxy per il download delle clip, con il risultato di sovraccaricare di meno il server che sta elaborando.

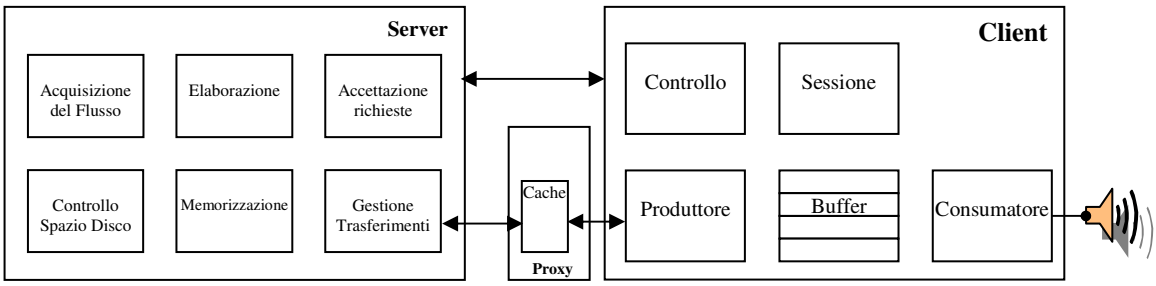


Figura 3.4 - Architettura del progetto

L'utilizzo del proxy verrà preso in considerazione dal server solo nel caso di più client: il proxy scarica le clip dal server e le rende disponibili ai client. In questo caso il numero di passaggi è superiore e il client dovrà attendere che il proxy scarichi dal server ciascuna clip. Il tempo di download sarà comunque molto basso considerando che il collegamento server-proxy sarà un collegamento di rete cablato veloce. Una volta sul proxy la clip potrà essere scaricata un numero arbitrario di volte senza che il server venga coinvolto. Con un solo client non avremmo benefici: aumenteremmo solamente il numero di passaggi e di conseguenza i ritardi.

3.3.3 Lato Client

Lato client, l'architettura dell'applicazione può essere riassunta dai blocchi riportati in figura 3.5. La componente principale che organizza l'applicazione è il blocco di controllo: si occupa della fase di inizializzazione e coordina il download e la riproduzione del media.

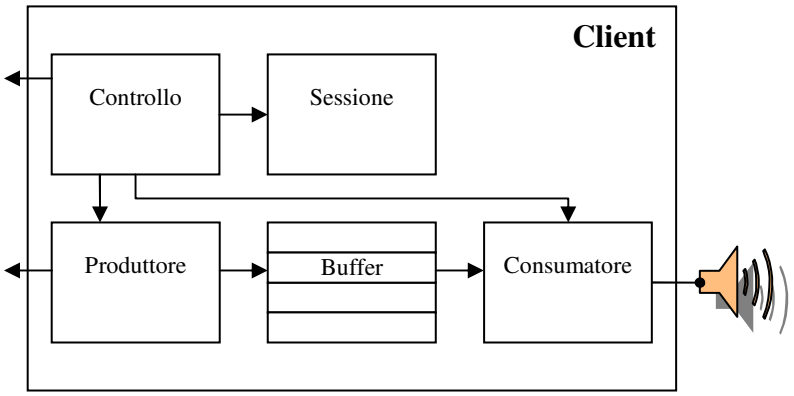


Figura 3.5 - Architettura del Client

Vediamo nel dettaglio i blocchi che costituiscono il client:

- *Controllo:* si occupa di gestire la richiesta al server, comunica con il produttore per avere informazioni sull'avanzamento della presentazione e in caso di interruzioni prolungate interroga il modulo di sessione. Ottiene dal server l'ok per iniziare la riproduzione, sia nel caso che la presentazione parta dalla sua richiesta, che nel caso di avvio precedente. In questo secondo caso viene specificato il nome e numero di clip attuale così che il client possa decidere da dove partire.
- *Sessione:* questo modulo di occupa di mantenere la sessione e in caso di interruzione del servizio decide da dove ripartire. Conosce la durata di ciascuna clip e il numero della clip attuale. Quando le condizioni di rete permettono la ripresa della trasmissione decide da dove continuare la riproduzione audio.
- *Produttore:* informato e controllato dal blocco di controllo questo modulo si occupa di reperire dalla rete le varie clip della presentazione e di scaricarle in un buffer. Può scaricare

direttamente dal server o da un proxy. Utilizza il protocollo HTTP per reperire i file.

- *Buffer:* permette la memorizzazione sul dispositivo mobile per fare fronte a cadute o degrado della comunicazione permettendo alla riproduzione di non subire discontinuità. Il buffer sarà condizionato dalle possibilità di memorizzazione del dispositivo mobile e del tempo di differita. Ad esempio se il server ha un anticipo di 15 secondi rispetto al client e le clip sono di 5 secondi ciascuna non è possibile scaricare più di 3 clip.
- *Consumatore:* Sfrutta la presentazione memorizzata sul buffer per eseguire la riproduzione delle clip audio utilizzando l'altoparlante del dispositivo mobile, eseguendo in successione le clip, per tutta la durata della presentazione.

Come descritto più volte il protocollo utilizzato per la comunicazione è HTTP. Questa scelta è stata fatta sia per la semplicità di utilizzo, sia perchè è implementato in tutte le piattaforme, contrariamente come può accadere ad esempio per le *socket*.
La parte che richiede un approfondimento ulteriore è il funzionamento dello schema produttore consumatore.
Il produttore acquisisce le varie clip audio in maniera sequenziale e le sistema in un buffer, il consumatore preleva dal buffer ed esegue la clip.
Il consumatore esegue le clip senza interruzioni finché il buffer non si svuota.
Le operazioni svolte vengono eseguite in parallelo: e necessitano di essere opportunamente sincronizzate.
Il produttore sarà vincolato dalla capacità del buffer e dal massimo numero di clip che possono essere scaricate. Questo numero dipende dalla durata di ciascuna clip e dal tempo di differita dal server.
Il produttore che riempie il buffer garantisce al consumatore una maggiore autonomia alimentandolo quando il collegamento non permette il download.

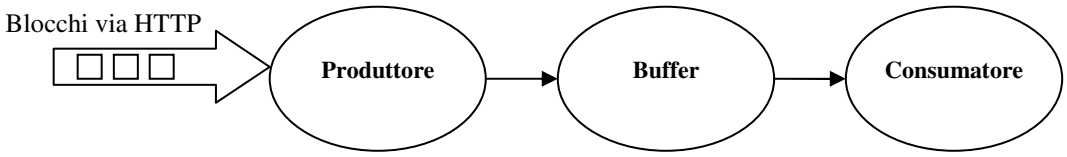


Figura 3.6 - Sincronizzazione dei due moduli lato client

La riproduzione in ordine delle clip audio richiede al buffer un funzionamento in modalità FIFO.
E' da sottolineare che la continuità di riproduzione può avvenire solo se il tempo di scaricamento di ciascuna clip richiede un tempo inferiore alla sua riproduzione. Con il tipo di file audio da noi impostato e con un collegamento Bluetooth questo è garantito.

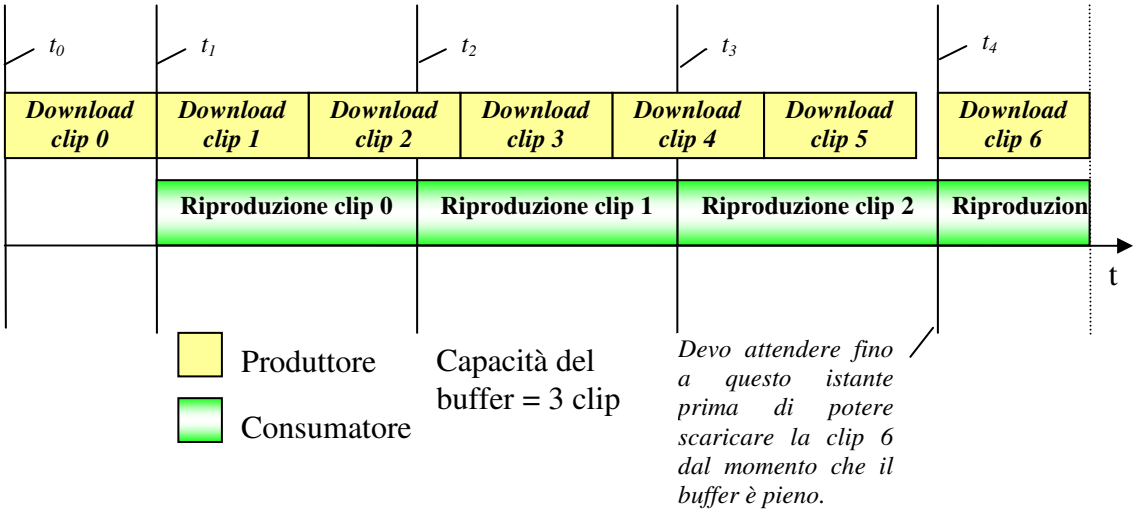


Figura 3.7 - Rappresentazione temporale Produttore-Consumatore con capacita limitata del buffer a 3 clip

Come mostrato in figura 3.7 il produttore continua a scaricare le presentazioni finché il buffer non è riempito. Quando il produttore termina il download della clip 5 la clip 6 non può essere scaricata perché il buffer contiene le clip 3, 4 e 5. Al tempo t_4 la clip 3 viene presa dal buffer ed eseguita e la clip 6 può essere scaricata.
Con un buffer più ampio il download continuerebbe come in figura 3.8.

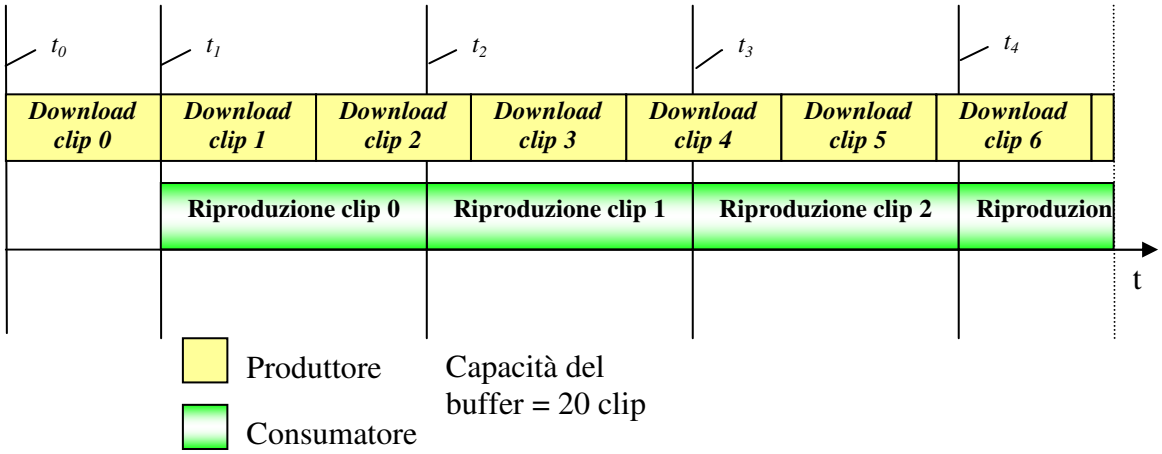


Figura 3.8 - Rappresentazione temporale Produttore-Consumatore con capacita limitata del buffer a 20 clip

La frammentazione, cioè la creazione delle varie clip viene fatta lato server, dove la maggiore capacità di risorse di elaborazione e memorizzazione permette di usare strumenti diversi e più potenti.
Lato client viene effettuato continuamente il download e riproduzione del media facendo come abbiamo detto, il merging delle tracce. Questa scelta è stata adottata per due motivi:

- *Capacità limitate dei dispositivi mobili*: le capacità di elaborazione e memorizzazione non permettono l'utilizzo di contenuti multimediali eccessivamente grandi. La riproduzione di un file richiede ad esempio di essere completamente scaricato localmente prima di potere essere riprodotto, diversamente da streaming, in cui la riproduzione avviene mentre si scaricano le parti successive.
- *Possibilità di degrado delle prestazioni di rete*: il collegamento wireless è soggetto a forti variazioni di prestazioni e al limite interruzioni. Se le interruzioni non sono troppo lunghe possiamo avere comunque continuità del servizio se si riesce a riscaricare la porzione successiva a quella attualmente in riproduzione prima del suo termine.

Questa nostra scelta permette anche di avere una certa qualità di servizio. Si vuole garantire che l'utente abbia sempre una corretta e sequenziale

visualizzazione della risorsa anche a fronte di interruzioni del collegamento wireless.

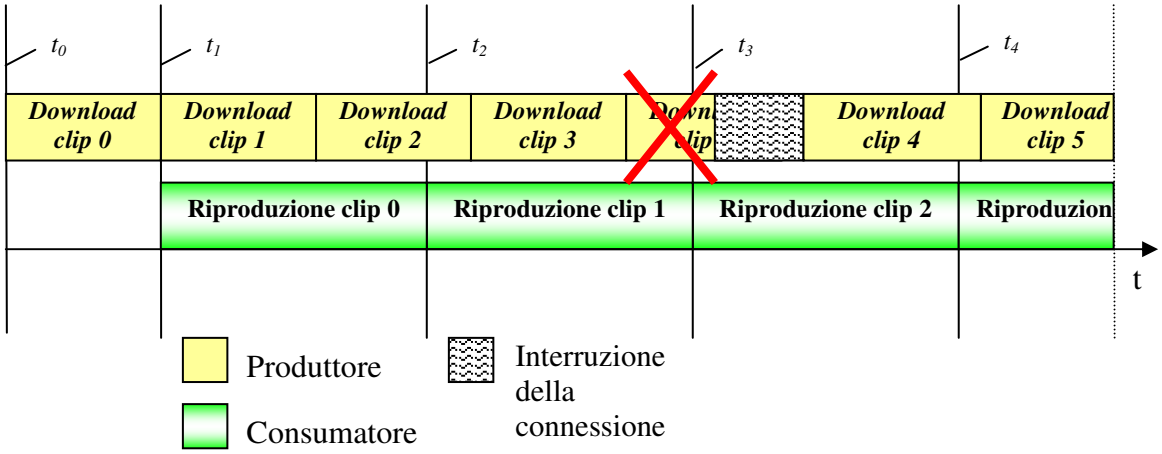


Figura 3.9 - Meccanismo di download e riproduzione lato client con interruzioni di breve entità

La presentazione appare come uno streaming, ma in realtà è la riproduzione in successione di tanti porzioni eseguite una di seguito all'altra.

3.4 Fasi di comunicazione

La comunicazione tra dispositivo mobile e server web avviene con collegamento bluetooth.
Nella nostra architettura ci sono due tipi di richieste effettuate dal client che utilizzano sempre il protocollo HTTP:

- *Richiesta di inizio servizio.* E' la richiesta del client che mette in azione l'elaborazione lato server. In questa fase il client invia i parametri al server, quali il formato in cui vuole ricevere i dati, i nomi delle varie clip, la durata di ogni clip. Il server controlla i parametri passati e la disponibilità a fornire il servizio. Al client viene notificata la disponibilità della presentazione e può iniziare a scaricarsela.
- *Richiesta di download di una clip*. Dopo l'esito positivo alla richiesta di inizio servizio, si può procedere alla richiesta di

download della clip. Questo significa che il server sta elaborando il flusso e sta creando le varie clip, nel formato specificato. E' quindi possibile procedere al download di ciascuna clip.

Per ogni clip, verrà eseguita una connessione HTTP. Per il meccanismo che abbiamo realizzato, verrà aperta una connessione per ogni clip. Le connessioni saranno sequenziali.

Nel caso di interruzioni di entità rilevante, il client può decidere se riprendere la riproduzione dall'ultima clip non ricevuta oppure da una clip particolare, dal momento che conosce la durata di ciascuna clip, il meccanismo di numerazione delle clip, la durata dell'interruzione della comunicazione.

3.5 Conclusione

In questo capitolo abbiamo analizzato l'architettura del nostro sistema, andando a definire i componenti principali che caratterizzano la nostra applicazione.

Il contesto di utilizzo può essere uno scenario di rete come Internet, il dispositivo mobile si collega in modalità wireless Bluetooth.

Abbiamo analizzato la parte server e client del progetto, ponendo attenzione alle fasi di comunicazione e descrivendo il protocollo utilizzato. Nel capitolo successivo parleremo di Java e di strumenti per lavorare con i media, questo nell'ottica di affrontare l'implementazione nel capitolo 5.

CAPITOLO 4

Java 2 Micro Edition, Mobile Media API e Java Media Framework

In questo capitolo dopo una breve presentazione delle varie distribuzioni Java esamineremo la distribuzione J2ME, soffermandoci sull'architettura di un'applicazione basata su di essa, in particolar modo vedremo come deve essere realizzata una MIDLet e il suo ciclo di vita.
Approfondiremo il discorso parlando di MMAPI e del supporto java ai media fornito dal Java Media Framework andando ad elencare i principali componenti necessari alle più importanti operazioni su flussi multimediali e confrontando poi le Mobile Media Application Program Interface con il Java Media Framework.

4.1 Famiglia Java

Nasce nel 1991 da un gruppo di ricerca per l'azienda californiana Sun Microsystem allo scopo di creare una tecnologia sulla quale l'industria dell'elettronica (televisori, elettrodomestici, altri piccoli dispositivi, ecc.) poteva fare affidamento. L'obiettivo era quello di sviluppare un software platform-independent, semplice da utilizzare.
Nel corso degli anni il linguaggio Java ha modificato i suoi obiettivi rilevando una grande diffusione, prima sul lato client, dovuto al supporto alla realizzazione di applet, poi sul lato server, grazie al supporto di tecnologie servlet (di cui Tomcat è un valido esempio) ed Enterprise Java Beans che hanno permesso la realizzazione di servizi WWW e sistemi back-end.

4.1.1 Le distribuzioni di Java

Con il rilascio di Java2 sono state introdotte le "edition" che raggruppano piattaforme, API e strumenti di sviluppo, indirizzati verso uno specifico settore di mercato. Ciascuna platform è basata su una Java Virtual Machine che che si appoggia sull'hardware sottostante.

- *Java 2 Platform, Standard Edition (J2SE)*, fornisce un ambiente per lo sviluppo e il supporto di applicazioni destinate a workstation e pc desktop.
- *Java 2 Platform, Enterprise Edition (J2EE)*, fornisce un ambiente basato su J2SE in cui sono state aggiunte API per la realizzazione di architetture scalabili, rivolto prevalentemente ad applicazioni server.
- *Java 2 Platform, Micro Edition (J2ME)*, è un insieme di tecnologie e specifiche rivolte a dispositivi embedded con risorse limitate, come telefoni cellulari, personal digital assistants (PDA), stampanti, tv set-top box.
- *Java Card technology* adatta per smart cards e altri dispositivi con memoria e capacità di calcolo limitata.

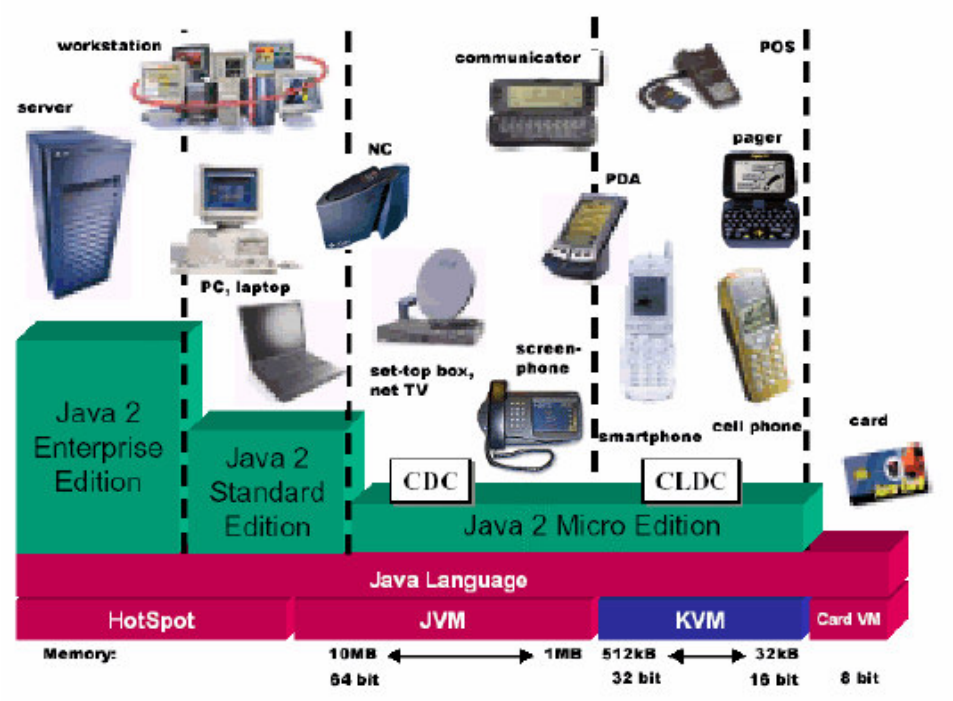


Figura 4.1 - Distribuzioni Java2

4.2 Piattaforma J2ME

Java 2 Micro Edition (J2ME) è la piattaforma Java per piccoli dispositivi, dalle risorse limitate. Diversamente dai sistemi server e desktop a cui si

rivolgono J2EE e J2SE, troviamo un elevato numero di dispositivi con caratteristiche diverse che rende impossibile creare un unico software che possa funzionare su tutti.
J2ME è quindi una collezione di specifiche che definisce in insieme di piattaforme, ciascuna delle quali è adatta ad un sottoinsieme di dispositivi. Il sottoinsieme di librerie Java che estende le funzioni base di una configurazione è definito da uno o più *profili*. La configurazione e profilo/i che si possono associare dipendono sia dal tipo di hardware del dispositivo che dal mercato a cui è rivolto. Si è cercato di ottimizzare principalmente l'utilizzo della memoria, del processore e tenendo conto anche delle capacità limitate di I/O di questi elementi.

4.2.1 Architettura di un'applicazione J2ME

L'architettura di un'applicazione sviluppata su piattaforma Java 2 Micro Edition risulta essere leggermente più complessa rispetto ad una normale applicazione: entrano in gioco i concetti di *configurazione* e *profilo*.

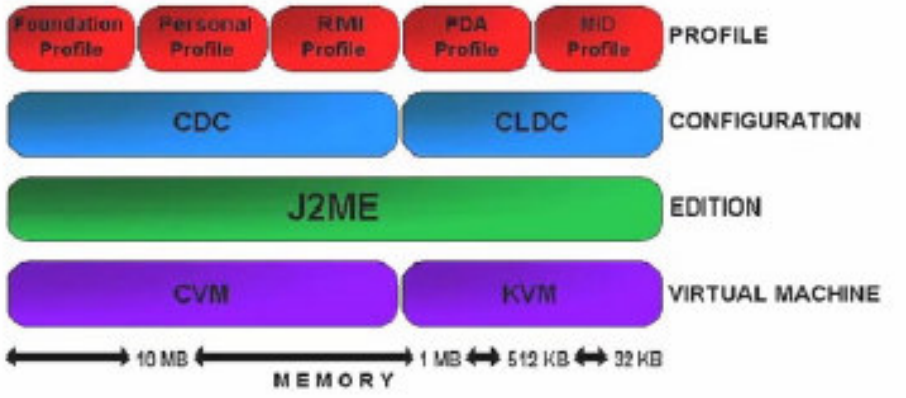


Figura 4.2 - Architettura J2ME

Per sistemi che aderiscono allo standard CDC, il sistema operativo si interfaccerà con la Java Virtual Machine classica, mentre nel caso di CLDC sarà la KVM o una qualsiasi Java Virtual Machine conforme alle specifiche. Sopra questa virtual machine si appoggiano le librerie di *Configurazioni* su cui appoggiano le API dei Profili. Ci sono inoltre packages opzionali utilizzabili dal programmatore per problemi specifici. Attualmente la piattaforma J2ME viene impiegata prevalentemente nella

programmazione di sistemi CLDC che sfruttano il profilo MIDP, cioè telefoni cellulari, computer palmari e cercapersone.

4.2.1.1 Concetto di Configurazione

Una configurazione è una specifica che definisce l'ambiente software per un insieme di dispostivi contraddistinti da caratteristiche comuni, come :

- Il tipo e la quantità di memoria disponibile
- Il tipo di processore e la velocità
- Il tipo connessione di rete disponibile al dispositivo

Una configurazione rappresenta la piattaforma minimale del dispositivo senza caratteristiche opzionali. I dispositivi devono supportare pienamente una configurazione, in questo modo è possibile creare applicazioni indipendenti dal dispositivo.

J2ME attualmente definisce 2 configurazioni

- *Connected Limited Device Configuration (CLDC):* è rivolta a dispositivi con caratteristiche limitate, come ad esempio cellulari o PDA con 512 KB di memoria disponibile. Per questa ragione, CLDC è strettamente legato al wireless Java, con la possibilità di scaricare piccole applicazioni Java, che prendono il nome di MIDlets.
- *Connected Device Configuration (CDC):* è rivolto a dispositivi che si trovano a metà strada tra i CLDC e i sistemi desktop con J2SE. Questi dispositivi hanno più memoria (2 MB o più) e processori più veloci, possono quindi supportare un ambiente più complesso. CDC si può applicare a PDAs e smart phones, gateways, set-top boxes.

Ciascuna configurazione comprende una Java Virtual Machine e una collezione di classi Java che forniscono l'ambiente per lo strato software applicativo. Le forti limitazioni di memoria di alcuni dispositivi possono limitare le caratteristiche di una J2ME Virtual Machine non supportando tutte le funzioni di una J2SE VM. Ad esempio dispositivi CLDC spesso non hanno il supporto hardware per operazioni a virgola mobile e in una CLDC VM non è richiesto il supporto ai tipi Java float e double o ogni metodo o classe che utilizza operazioni in virgola mobile.

Dove è possibile, J2ME deve riusare classi a packages di J2SE.

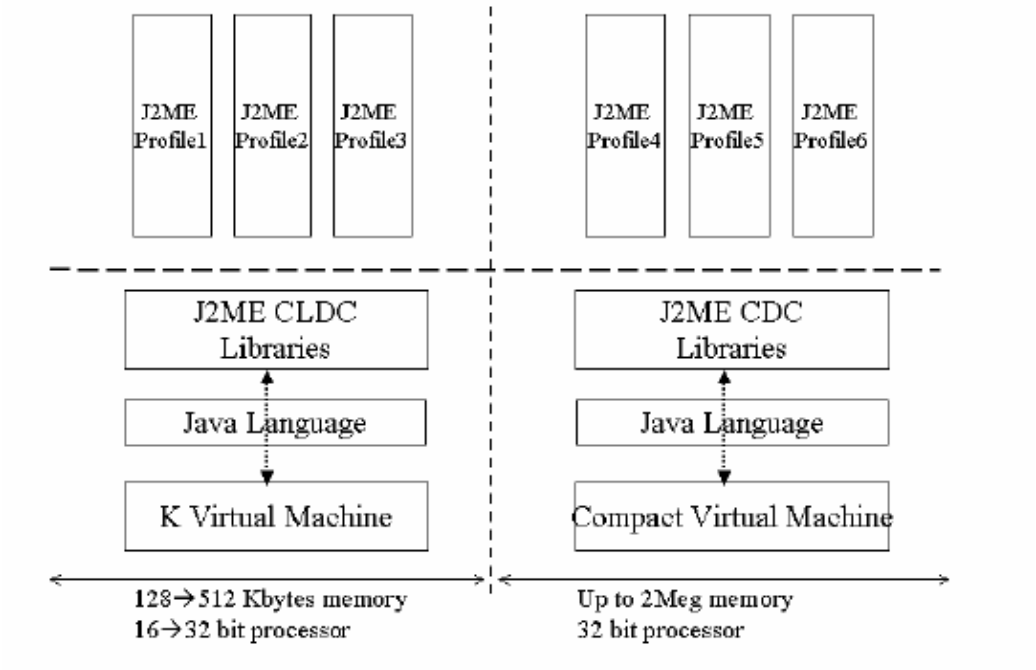


Figura 4.3 - Struttura CDC, CLDC

4.2.1.1 Concetto di Profilo

Un profilo completa la configurazione aggiungendo nuove classi che forniscono caratteristiche aggiuntive. Il programmatore può quindi scrivere applicazioni rivolte a dispositivi particolari o per specifici settori di mercato.

Le due configurazioni J2ME hanno uno o più profili associati, alcuni dei questi possono fare a loro volta uso degli altri profili. La figura 4.4 mostra i profili che sono attualmente definiti e le configurazioni su cui si basano.

Vediamoli nel dettaglio:

- *Mobile Information Device Profile (MIDP)*: Questo profilo aggiunge componenti di rete, interfaccia utente e memorizzazione locale al CLDC. Questo profilo prende in considerazione le limitazioni dello schermo e della memoria del dispositivo mobile, fornendo un'interfaccia semplice e funzionalità di rete basate sul protocollo HTTP 1.1. MIDP è il più conosciuto tra i profili perché

è la base per il wireless Java ed è l'unico profilo per portatili con PalmOS.

- *PDA Profile (PDAP)*: Il PDA Profile è simile al MIDP, ma è rivolto a PDA, che hanno uno schermo migliore è più memoria di un cellulare. Il PDA Profile offre una libreria di interfaccia utente più sofisticata e delle API per un uso più comodo delle funzioni del sistema operativo
- *Foundation Profile*: Questo profilo estende il CDC includendo la maggior parte dai packages Java 2 versione 1.3. E' molto importante perchè e nato per essere la base degli altri profili CDC.

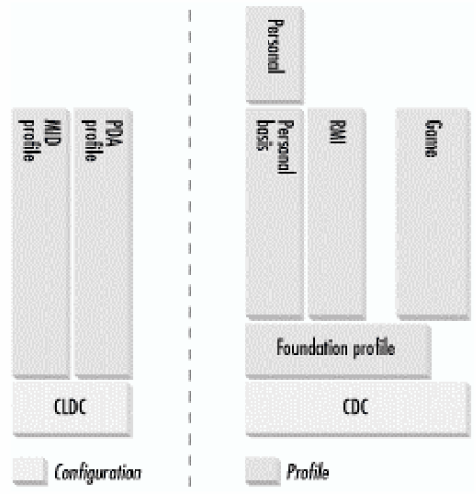


Figura 4.4 - Configurazioni e profili J2ME

- *Personal Basis and Personal Profiles*: Il profilo Personal Basis aggiunge le funzionalità di base per l'interfaccia utente al Foundation Profile. E' rivolto a dispositivi con display semplici che non permette a più di una di finestra di essere attiva nello stesso momento. Piattaforme che supportano un'interfaccia utente più complessa invece useranno il Personal Profile.
- *RMI Profile*: Questo profilo aggiunge le librerie di Remote Method Invocation di J2SE al Foundation Profile, supportando solo il lato client.
- *Game Profile*: Il Game Profile fornisce il supporto per la scrittura di giochi su dispositivi CDC. Questi dispositivi dispongono di

elevata memoria, con 32-64 MB di RAM grafica 3D accelerata. Anche se sono dispositivi con buone prestazioni si è preferito utilizzare J2ME piuttosto che J2SE.

4.2.2 Connected Limited Device Configuration (CLDC)

La Connected Limited Device Configuration (CLDC) è il blocco di base su cui si appoggiano i profili J2ME per dispositivi di piccole dimensioni, quali cellulari e PDA con risorse limitate. Date le peculiarità di questi dispositivi risulta impensabile che possano supportare una piattaforma Java come J2SE. CLDC specifica un insieme dei packages e classi minimali, con una Java Virtual Machine ridotta. Basti pensare che la semplice applicazione "Hello World" su un sistema Windows richiede un'allocazione di memoria di circa 16 MB.
Le caratteristiche minime per il supporto CLDC sono:

- 128 KB di ROM, flash o altro tipo di memorizzazione persistente della Java VM e delle librerie di classi che costituiscono la piattaforma CLDC.
- 32 KB o più di memoria volatile per l'allocazione runtime. Questa memoria è utilizzata dalla Java VM per richieste dinamiche, come il caricamento delle classi e l'allocazione dello spazio heap e stack.

Oltre che sui requisiti di memoria, CLDC fa poche assunzioni. Ad esempio non assume che il dispositivo debba avere un certo tipo di display o strumenti di input come tastiera o mouse, e non richiede nessun tipo di memorizzazione locale per le applicazioni. Minimizzando i requisiti CLDC si massimizza il numero di piattaforme su cui può essere implementato. Si assume solo che il dispositivo abbia un sistema operativo che possa eseguire e gestire la JVM. Benché Java sia un ambiente multithreaded, non è necessario per il sistema operativo avere il concetto di thread o di essere in grado di schedulare più di un processo alla volta.

4.2.2.1 Caratteristiche della Virtual Machine e del linguaggio

Le specifiche CLDC definiscono le caratteristiche che una VM deve avere descrivendo le specifiche della Java Virtual Machine completa e delle specifiche del linguaggio Java di cui *non* è richiesto il supporto e i punti in

cui queste limitazioni sono applicabili. Sun fornisce una implementazione di riferimento delle specifiche CLDC che è basata sulla KVM, che è una virtual machine che soddisfa appunto i requisiti CLDC. In generale può essere usata ogni virtual machine che soddisfa le specifiche CLDC.

4.2.2.2 Limitazioni

Vediamo quali caratteristiche del linguaggi non sono supportate in un ambiente di specifiche CLDC, confrontando le differenze con J2SE.
La maggior parte del processori usati nei dispositivi in cui utilizziamo CLDC non hanno il

- *supporto al calcolo in virgola mobile*, di conseguenza alla virtual machine non è richiesto di supportare operazioni di questo tipo. Questo significa che non sono implementate le seguenti operazioni:

Dadd	dload	dsub	fcmpl	frem	i2d
Daload	dload_x	d2f	fconst_0	freturn	i2f
dastore	dmul	d2i	fconst_1	fstore	l2d
dcmpg	dneg	d2l	fdiv	fstore_x	l2f
dcmpl	drem	fadd	fload	fsub	newarray (double)
dconst_0	dreturn	faload	fload_x	f2d	newarray (float)
dconst_1	dstore	fastore	fmul	f2i	
ddiv	dstore_x	fcmpg	fneg	f2l	

Tabella 4.1 - Operazioni in virgola mobile non implementate da una CLDC VM

Questo porta alle seguenti restrizioni di codice

- Le variabili di tipo *float* e *double* e array di questi tipi non possono essere dichiarati o usati.
- Costanti di tipo *float* e *double* non possono essere usate.
- Gli argomenti dei metodi non possono essere di tipo *float* o *double*.
- I metodi non possono avere come valore di ritorno valori di tipo *double* o *float*.
- Oggetti di tipo *Float* e *Double* non possono essere creati.

Sun non fornisce una versione diversa del compilatore da usare quando si sviluppa una applicazione CLDC, dunque è possibile, usando un compilatore J2SE creare classi che usano i tipi in virgola mobile violando

queste regole. Tuttavia queste classi non saranno accettate durante la verifica dei file class dalla CLDC virtual machine.
Oltre a queste limitazioni troviamo altre omissioni nel linguaggio:

- *Reflection*: Il package java.lang.reflect e tutte le caratteristiche di java.lang.Class che sono collegate alla reflection non sono disponibili. Questo in parte per risparmiare memoria, ma anche per risparmiare memoria nel controllo dei privilegi per l'accesso a queste caratteristiche.
- *Weak references*: Weak references e il package java.lang.ref non sono supportati a causa della memoria richiesta per la loro implementazione.
- *Object finalization*: rendere un Oggetto final provoca una grande complessità nella VM con pochi benefici in cambio. Perciò non viene implementata, e la classe CLDC java.lang.Object non ha il metodo finalize().
- *Threading features*: CLDC supporta threads, ma non permette la creazione di un demone thread (un thread che è automaticamente terminato quando tutti gli altri threads nella VM terminano) o gruppi di threads. Se si vogliono gestire delle operazioni su un gruppo di thread, occorre utilizzare esplicitamente una collezione di oggetti a livello applicativo per gestire l'oggetto thread come gruppo.
- *Errori ed Eccezioni*: J2SE ha un gran numero di classi che rappresentano errori e condizioni di eccezioni. Dal momento che in generale non si possono gestire gli errori, la maggior parte delle classi che rappresentano errori non sono incluse nella piattaforma CLDC. Quando si verifica un errore il dispositivo esegue un'azione appropriata invece di riportarlo al programma.
- *Java Native Interface*: CLDC non fornisce le caratteristiche JNI di J2SE, che permettono al codice nativo di essere chiamato da classi Java. JNI è omesso sia per l'uso intensivo di memoria necessaria ad implementarlo, sia per la memoria necessaria ad implementare strumenti di sicurezza.

La virtual machine che supporta il CLDC è realizzata per identificare i file .class non validi. Questa operazione è molto costosa. L'operazione di pre-verifica viene fatta fuori dal dispositivo, ad esempio su una stazione server. Una volta che l'operazione di pre-verifica si è conclusa correttamente, il file .class viene caricato sul dispositivo.

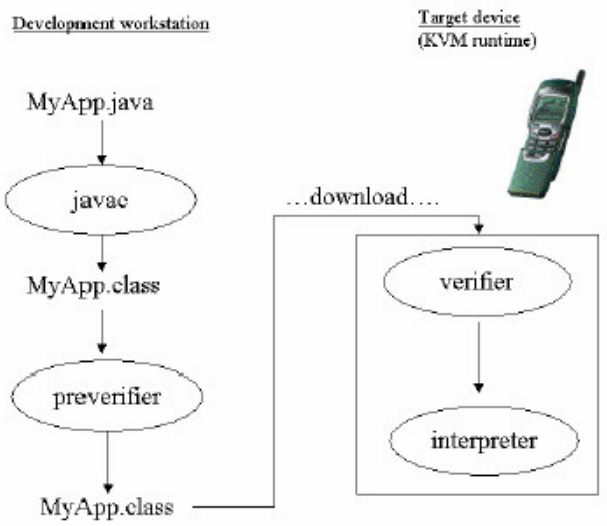


Figura 4.5 - Pre-verifica file .class in CLDC/KVM

4.2.2.3 Sicurezza

In J2SE il modello di sicurezza è abbastanza potente da permettere a codice originato da diverse fonti di avere diversi livelli di privilegio e di conseguenza diversi livelli di accesso alle risorse di sistema. Le applicazioni installate su una macchina utente, di default non hanno limitazioni. Un applet scaricato da web opera in un ambiente ristretto che non gli permette di accedere alle risorse locali come file system e limita l'accesso alla rete. La sicurezza permette dei privilegi che devono essere assegnati o negati ad un'applicazione o ad un applet basandosi sulla veridicità dell'origine. Il codice deve essere accompagnato da un certificato che garantisce la provenienza e attendibilità. Può essere anche firmato facendo uso degli algoritmi di crittografia, in modo che il ricevente possa essere sicuro che non sia stato modificato.
Una virtual machine CLDC può essere usata in un dispositivo che non permette all'utente di installare programmi, perciò ha necessità di minori misure di sicurezza.

Potrebbe anche essere usata in un telefono cellulare connesso alla rete per il download di applicazioni. La rete dovrebbe essere soggetta allo stesso tipo di ambiente di sicurezza che si applica alle applets J2ME. Sarebbe utile avere livelli intermedi di sicurezza per il codice proveniente da una fonte attendibile. Questo non sempre è possibile perché una struttura sulla falsa riga della J2SE per la sicurezza richiede un utilizzo cospicuo di memoria che non è applicabile su un dispositivo CLDC. La virtual machine CLDC fa eseguire le applicazioni in un ambiente noto come "sandbox" il quale assicura che non possa danneggiare il dispositivo su cui viene eseguita.

4.2.3 Mobile Information Device Profile (MIDP)

La Connected Limited Device Configuration fornisce le basi per lanciare Java in dispositivi che non hanno risorse adeguate per supportare una Java Virtual Machine completa di tutti i packages della distribuzione J2SE. Tuttavia è improbabile che uno sviluppatore utilizzi solamente le API fornite dalla CLDC, perché non forniscono nulla per l'interazione con l'utente, dispositivi di memorizzazione, o rete. Il Mobile Information Device Profile, è uno dei profili che si appoggia la CLDC e KVM, ed è rivolto a dispositivi con interfaccia utente limitata a un piccolo display con poche funzionalità di input come ad esempio cellulari. E' indicato anche per PDAs e palmOS dalla versione 3.5.

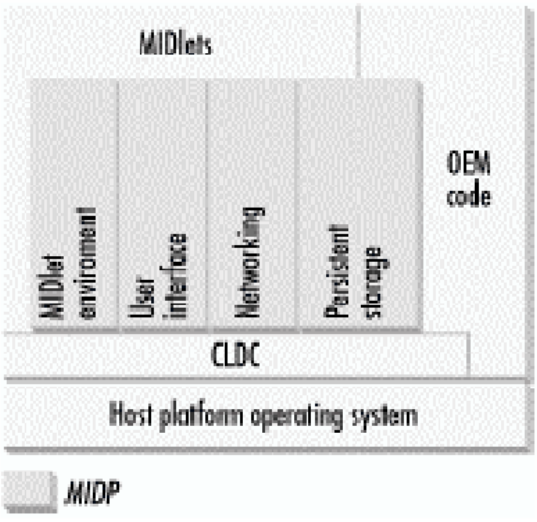


Figura 4.6 - Il Mobile Information Device Profile

Le specifiche MIDP sono state sviluppate sotto il Java Community Process, per maggiori dettagli si veda [MIDP].
La struttura logica della posizione di MIDP nell'architettura di un dispositivo che lo implementa è mostrato in figura 5.
Il software che utilizza MIDP usa la KVM resa disponibile dal CLDC e fornisce ulteriori funzionalità che possono essere utilizzate dal programmatore utilizzando le API MIDP. Le applicazioni MIDP vengono chiamate MIDlets. Come mostrato in figura una MIDlet può sia utilizzare le API MIDP che le API fornite dal CLDC. Dato che KVM non supporta JNI, l'unica via per un'applicazione MIDlet per accedere alle caratteristiche a basso livello è di utilizzare api specifiche di una virtual machine personalizzata.
Sun fornisce un'implementazione di MIDP che può essere usata sotto Windows: è il Wireless Toolkit, che contiene la versione MIDP per Windows, Solaris e Linux.
Alcune parti del core MIDP sono a volte come i servizi CLDC dipendenti dal dispositivo e così sono forniti dal produttore, tipicamente includono funzionalità per il supporto di rete, componenti per l'interfaccia utente e la memorizzazione locale.

4.2.3.1 Requisiti hardware

Come più volte accennato , MIDP è rivolto a dispositivi con memoria, CPU e display con funzionalità limitate. I requisiti minimi sono:

- *Memoria*: Le specifiche MIDP richiedono almeno 128 KB di RAM per l'implementazione MIDP stessa oltre a quella richiesta per CLDC, in aggiunta ne occorrono 32 KB per l'heap Java. Data la ridotta dimensione dell'heap è necessario deallocare gli oggetti appena non sono più necessari affinché il garbage collector possa liberare memoria. Sono richiesti inoltre altri 8KB di memoria non volatile per la memorizzazione di informazioni da parte delle MIDlets.
- *Display*: Dispositivi MIDP sono caratterizzati da piccoli display, con dimensioni minime di 96 pixel altezza e 54 pixel di larghezza,

con pixel approssimativamente quadrati. Lo schermo deve avere almeno 2 colori.

- *Dispositivi di input*: come per il display, ci sono diversi tipi di dispositivi di input che possono essere utilizzati, ma l'assunzione minima è di avere un tastierino con numeri da 0 a 9, l'equivalente di frecce per la navigazione, un bottone di selezione . Come esempio di veda la figura 4.6.

Figura 4.7 - Tipica tastiera di un cellulare

- *Connettività*: Esistono diverse modalità di accesso alla rete, come connessione wireless o con un modem aggiuntivo per un PDA. MIDP non suppone che il dispositivo sia sempre collegato alla rete o che la rete supporti il TCP/IP. Tuttavia è richiesto che ci sia il supporto per HTTP 1.1, o direttamente sullo stack di protocolli internet o utilizzando una connessione wireless mediante un gateway WAP.

4.2.3.2 Requisiti hardware

Come accennato, le specifiche MIDP di SUN non sono un prodotto commerciale, ma ci saranno diverse implementazioni, personalizzate a seconda dell'hardware. Il sistema operativo deve dare un ambiente di supporto alla Java Virtual Machine. Dato che il CLDC supporta i thread, idealmente la piattaforma supporta il multithreading e la KVM può utilizzarlo.

E' richiesto il supporto di rete, su alcune piattaforma è disponibile il livello socket, in ogni caso deve essere supportato l'HTTP. Nel nostro

applicativo è stato utilizzato l'HTTP per comunicare da dispositivo a server.
Il software deve anche fornire l'accesso al dispositivo di input e un dispositivo di puntamento se disponibile. Deve essere anche in grado di gestire gli eventi, alla pressione di un tasto.
Deve essere possibile memorizzare in maniera permanente, per questo sono disponibili delle API apposite.

4.2.4 MIDlet

Una MIDlet è un'applicazione Java che si appoggia su un dispositivo in cui è installato MIDP. Una MIDlet è composta da almeno una classe Java che deriva dalla classe astratta *javax.microedition.midlet.MIDlet*. Le MIDlets vengono eseguite all'interno della VM, seguendo un ciclo di vita controllato attraverso i metodi della classe *MIDlet* implementata.

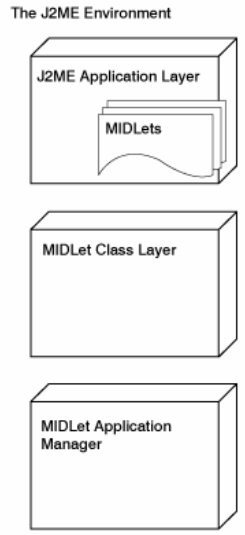


Figura 4.8 - Livelli dell'ambiente J2ME

Un insieme di MIDlet può essere raggruppato in una MIDlet suite. Tutte le MIDlet in una suite fanno parte di un package e vengono installate ed eventualmente disinstallate sul dispositivo come singola entità. Le MIDlets appartenenti ad una suite condividono risorse statiche e dinamiche dell'ambiente di esecuzione come:

- A runtime , se il dispositivo supporta applicazioni concorrenti tutte le MIDlets appartenenti ad una MIDlet suite eseguono nella stessa

JVM. Questo significa che i dati possono essere condivisi attraverso le MIDlets e le primitive di sincronizzazione possono essere usate per gestire la concorrenza dentro una singola MIDlet, ma anche tra diverse MIDlets della stessa suite.

- La memoria permanente è gestita a livello di MIDlet suite, quindi le MIDlets di una suite condividono lo stesso spazio. Ciò non è vero per MIDlets appartenenti a suite diverse.

4.2.4.1 Sicurezza di una MIDlet

Il modello di sicurezza applicato in J2SE è molto potente, ma allo stesso tempo costoso in termini di risorse. Per questo motivo non ci sono meccanismi di sicurezza per le MIDlets. E' necessario quindi assicurarsi che le MIDlets provengano da fonti sicure e che non siano state modificate. Il protocollo HTTPS potrebbe essere una soluzione. Comunque non ci sono API MIDlet che hanno accesso a informazioni personali come rubrica o agenda.

4.2.4.2 Packaging di una MIDlet

Le MIDlets devono essere inserite in un package per potere essere eseguite. Ci deve essere la classe principale, che è sottoclasse di *MIDlet*, con le classi e file di cui ha bisogno e devono essere inserite in un singolo file JAR (Java Archive). Un JAR può contenere più di una MIDlet, in questo caso si parla di suite.
All'interno del file JAR è presente il file *manifest* che è un file di testo in cui è descritto il contenuto dell'archivio attraverso una serie di attributi e rispettivi valori separati da ":":
nome-attributo: valore-attributo
Tutti gli attributi che sono rilevanti per l'installazione della MIDlet hanno come parte iniziale del nome "MIDlet-".
Informazioni simili sono fornite anche dal Java application descriptor o JAD, che è separato dallo JAR. Viene utilizzato per potere essere scaricato più velocemente, dando informazioni sull'applicazione. Tali informazioni sono necessarie all'*Application Manager Software* (AMS) per verificare se la MIDlet è in grado di operare sul dispositivo e di procedere al download e installazione.

4.2.4.3 Esecuzione e ciclo di vita di una MIDlet

Tutte le MIDlets derivano dalla classe astratta *javax.microedition.midlet.MIDlet* che contiene metodi che la piattaforma MIDP chiama per controllare il ciclo di vita della MIDlet, la MIDlet stessa può usare questi metodi per cambiare il suo stato. La MIDlet si può trovare in uno dei tre stati: *Paused, Active, Destroyed*.

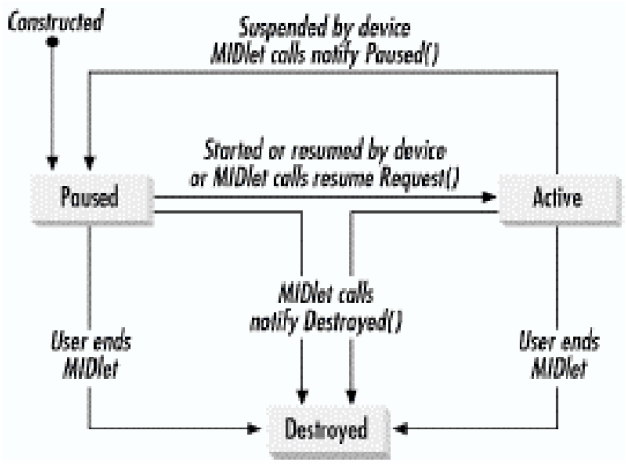


Figura 4.9 - Livelli dell'ambiente J2ME

Quando una MIDlet è caricata, inizialmente è nello stato *Paused*. Se durante l'esecuzione si verificano problemi, viene lanciata un'eccezione e si raggiunge lo stato Destroyed. Se lo stato passa da *Paused* ad *Active*, viene chiamato il metodo *startApp()*. Questo è un metodo astratto che deve essere implementato dalla MIDlet. Se non si riscontrano problemi, l'applicazione incomincia ad operare.

4.3 Mobile Media API

Mobile Media API (MMAPI) è una package Java per ambiente J2ME che permette ad un dispositivo di lavorare con contenuti multimediali, come clips audio, sequenze MIDI, filmati, e animazioni. I contenuti multimediali stanno diventando sempre più importanti per molte applicazioni, come nel nostro caso servizi on-online. Con le MMAPI possiamo realizzare diversi servizi quali:

- Servizi di informazione arricchiti con commenti video e audio

- Servizi di messaggistica arricchiti dove al testo sono aggiunti video e audio
- Giochi on-line interattivi che integrano funzionalità audio e video
- Cattura di immagini e relativa visualizzazione
- Stream video, audio e di clips animate
- Sistemi di sicurezza per il controllo di ambienti e sistemi di allerta-emergenza
- Comunicazioni interne
- Video/audio conferenze.

Il dispositivo mobile si avvicina quindi ad uno strumento con caratteristiche che ricordano quelle di un desktop, e diventa sempre più uno strumento personalizzabile, da utilizzare sia per lavoro che per svago.
Se i dispositivi wireless hanno sempre problemi di memoria, la capacità di processing diventa sempre più potente. Molti PDA di fascia alta sono paragonabili a pc desktop di pochi anni fa, da qui l'idea di lavorare con contenuti multimediali su questi dispositivi. In J2ME MIDlets che utilizzano queste funzionalità sono piccole (10-30KB) e possono essere scaricate velocemente occupando poca memoria.

4.3.1 Caratteristiche delle MMAPI

Le MMAPI sono un package opzionale della piattaforma J2ME, di conseguenza, rispettando le specifiche J2ME è stato implementato rivolgendo attenzione alle risorse necessarie alla corretta esecuzione: limitazioni di banda, problemi di traffico, connessione lenta, e bassa risoluzione degli schermi. Queste API permettono un accesso semplice al contenuto multimediale.
Le MMAPI fanno uso di quattro elementi:

- Un *Player* che prende e decodifica il media. Il Player è completamente neutrale al tipo di contenuto che riceve. Cioè non è necessario un player per video e un player per una traccia audio, la riproduzione dipende solo dai controlli che sono associati al Player.
- Il *tipo* di media che deve essere riprodotto dipende dai *controlli* che sono associati al Player. Per essere riprodotto, ciascun media richiede, o è comunque consigliato che venga associato al Player

uno o più controlli. Per esempio se di vuole riprodurre una traccia audio, bisognerebbe aggiungere il controllo per il volume e un controllo per impostare lo stop.

- Un *DataSource* fornisce il modo per trattare la sorgente multimediale, supportando meccanismi base per la riproduzione e sincronizzazione. Nasconde dettagli di come i dati siano letti dalla sorgente: un file, uno streaming internet da server, o un sistema di acquisizione proprietario. Un DataSource è completamente trasparente al meccanismo di reperimento, si può usare HTTP, un protocollo di streaming come Real-time Transport Protocol (RTP), o altri meccanismi.
- Il *Manager* ha la visione di ogni cosa, permettendo di creare il Player e associargli un DataSource. Al Manager può essere richiesto l'elenco dei tipi di media e protocolli supportati. Può anche riprodurre velocemente toni.

Questi sono gli elementi compongono un API. Valutando le MMAPI osserviamo che:

- MMAPI richiede poca occupazione di memoria: è progettato per essere eseguito su una Java Virtual Machine con funzionalità CLDC o superiori. La richiesta di memoria è di circa 20KB (non compresso).
- Le MMAPI sono neutrali al protocollo e contenuto: i componenti principali sono il Player e il DataSource. Il Player accetta semplicemente i dati, li decodifica e li riproduce, sullo schermo o sul dispositivo audio. Il contenuto può essere un media audio o video. Il Player non deve supportare particolari codec. I dettagli dei codec supportati sono lasciati al profilo (come ad esempio MIDP). Il DataSource incapsula la gestione del protocollo. Nasconde i dettagli di come i dati sono letti dalla sorgente. Il Datasource gestisce i dettagli della connessione: sia con connessione HTTP che RTP.
- Le MMAPI sono estendibili: il Player e il DataSource sono stati progettati per essere flessibili. E' sempre possibile estendere il

DataSource con protocolli personalizzati. Il Player legge dal DataSource personalizzato per riprodurre il media.

- Le MMAPI sono scalabili: possono essere usate sia su dispositivi di fascia alta che dispositivi di fascia bassa. Possono essere usate ovunque, dal CLDC a J2SE, ma sono comunque API rivolte al lato client. Per applicazioni J2SE o J2EE che girano su macchine con maggiori risorse si può usare al loro posto, ed è quello che è stato fatto, il Java Media Framework (JMF). Sia JMF che MMAPI sono neutrali al protocollo, così si può usare lato server il JMF e lato client le MMAPI utilizzando HTTP, RTP o altri protocolli.
- Le MMAPI supportano tutti i formati media time-based e non si rivolgono a un formato particolare: non essendo possibile sapere il tipo di dispositivo su cui saranno utilizzate le MMAPI, le API non garantiscono quali formati saranno effettivamente supportati. Le MMAPI permettono la libera implementazione di formati che un dispositivo può supportare. Per esempio, se venissero usate le MMAPI su un dispositivo karaoke, occorrerebbe implementare la cattura audio e la riproduzione, senza dovere supportare un formato video o controlli video.
- Le MMAPI sono progettate per utilizzare al meglio servizi multimediali disponibili nel codice nativo o nell'hardware di un certo dispositivo: in molti dispositivi la CPU non è abbastanza potente per potere affidare a un programma Java la decodifica di un media. Per esempio i cellulari hanno un'implementazione nativa di riproduzione audio o toni. In questo caso le MMAPI si appoggiano su questi servizi e le applicazioni Java accedono ai servizi a basso livello.

4.3.2 Player

Il Player controlla il rendering del media. Fornisce metodi per gestire il ciclo di vita del Player, ottiene i componenti necessari e controlla la riproduzione. Può essere creato tramite il metodo *createPlayer* della classe *Manager*. Un Player ha cinque stati: *UNREALIZED, REALIZED, PREFETCHED, STARTED, CLOSED*.

Lo scopo di questi stati è di fornire un controllo alle operazioni di gestione del Player. Quando un Player è costruito per la prima volta si trova nello stato *UNREALIZED*. Raggiunge lo stato *REALIZED* eseguendo tutte le comunicazioni necessarie per acquisire le risorse necessarie (comunicando ad esempio con un server o con il file system). Il metodo *realize* permette ad un'applicazione di portare il Player nello stato *REALIZED*.
Tipicamente un Player passa dallo stato *UNREALIZED* allo stato *REALIZED,* poi allo stato *PREFETCHED* ed infine allo stato *STARTED*. Un Player termina quando raggiunge la fine del media, o quando il metodo *stop* viene invocato. Quando ciò avviene il Player si porta dallo stato *STARTED* allo stato *PREFETCHED* ed è pronto a ripetere il ciclo.
Per utilizzare un Player è necessario impostare i parametri per gestire il passaggio attraverso il suo ciclo di vita utilizzando i metodi di transizione.

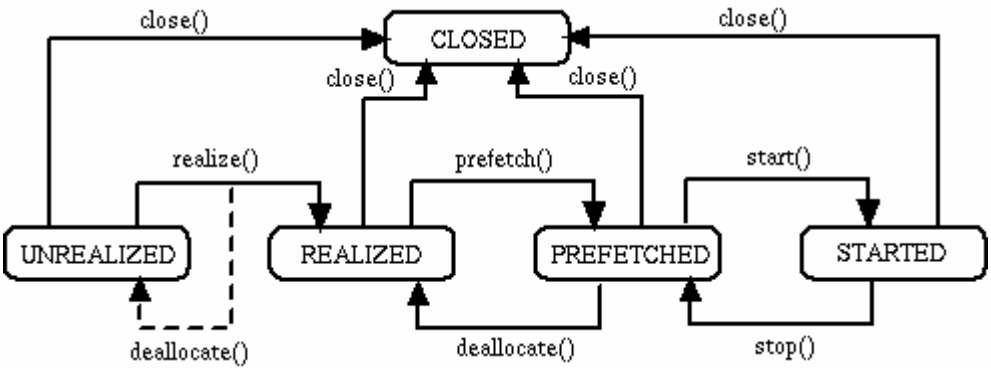


Figura 4.10 - Ciclo di vita di un Player

4.3.3 Manager

Il Manager è il punto di accesso per ottenere le risorse dipendenti dal sistema, come ad esempio un Player. Il Manager si occupa della gestione a basso livello dei Player, cioè gestisce l'allocazione e deallocazione delle periferiche (display, altoparlante, microfono, fotocamera, ecc.) e fornisce informazioni sull'hardware. Attraverso la classe Manager si può ottenere un'istanza di Player utilizzando il metodo *createPlayer* disponibile in tre diverse forme:

public static createPlayer(InputStream stream, String type)

public static createPlayer(String locator)
public static createPlayer(DataSource source)

La prima forma del metodo permette la creazione di un Player a partire da uno stream di input associato alla risorsa da riprodurre. Il secondo parametro passato (type) rappresenta il "content-type" della risorsa. Esempi di content type sono:

- Audio in fromato Wave: audio/x-wav
- Audio in formato AU: audio/basic
- Audio in formato MP3: audio/mpeg
- Audio in formato MIDI: audio/midi
- Sequenze di toni: audio/x-tone-seq
- Video in formato mpeg: video/mpeg

Con la seconda forma la stringa passata per creare il Player è una URI ed ha un preciso formato.
La terza forma costruisce il Player da un DataSource.
Il metodo *String[] getSupportedContentTypes(String protocol)* ritorna una lista di tutti i content-type suportati dal protocollo passato come parametro.
Il metodo *String[] getSupportedProtocols(String content_type)* restituisce una lista di protocolli supportati per il content type passato. Se viene passato null restituisce l'elenco di tutti i protocolli supportati dall'implementazione.

4.3.4 Datasource

Il media può essere reperito da sorgenti molto diverse tra loro, come ad esempio un file video su file system locale, tracce audio su un cd , streaming attraverso internet. La classe indicata per descrivere e gestire queste diverse sorgenti è la classe *DataSource*. La creazione di un Player, tramite la classe Manager prima necessita della creazione di un DataSource:

```
source = Manager.createDataSource(SourceLocation);
String tipo = source.getContentType();
```

Il DataSource fornisce un modello astratto del media e semplifica la gestione del trasferimento del contenuto del media. La creazione richiede la specifica del protocollo e il percorso del media. Mediante queste specifiche il Manager segue un processo iterativo che consiste nel cercare quel DataSource che supporta il protocollo e il formato specificati.
Una volta utilizzato un DataSource, non può essere riusato.
La creazione di un DataSource si ottiene con il metodo *createDataSource* che accetta un URL o un *MediaLocator* che rappresenta il percorso del media.
La classe astratta DataSource rappresenta l'astrazione dei protocolli di basso livello che gestiscono il trasferimento dei dati multimediali.
DataSource e SourceStream hanno funzionalità non presenti nei normali stream (InputStream), come l'accesso random ai dati e la possibilità di suddividere il flusso in "blocchi". Un DataSource, implementando l'interfaccia *Controllable*, può essere controllato attraverso opportuni controlli ottenibili con *getControl* e *getControls*.
Il trasferimento dei contenuti multimediali può essere gestito attraverso i seguenti metodi:

- *public abstract void connect()*
- *public abstract void start()*
- *public abstract void stop()*
- *public abstract void disconnect()*

Il primo apre una connessione con la sorgente specificata dal parametro passato al costruttore della classe DataSource. Con la *start* si può avviare il trasferimento dei dati che può essere interrotto con il metodo *stop*. Infine per rilasciare la connessione aperta viene invocato il metodo *disconnect*.
Sono previsti anche i metodi

- *public abstract String getContentType()*
- *public String getLocator()*

che ritornano rispettivamente il content type del media restituito dal DataSource e il locator associato al DataSource stesso.

4.3.1 Control

Un oggetto *Control* è usato per controllare alcune funzioni di esecuzione dei media. Control è ottenuto da *Controllable*; è l'interfaccia Player che estende Controllable. L'implementazione di un Player può usare l'interfaccia Control per estendere le funzioni di controllo del media. Per esempio un Player può esporre un *VolumeControl* per permettere la regolazione del volume.
Possono essere implementati *Control*s multipli dallo stesso oggetto. Ad esempio, un oggetto può implementare sia *VolumeControl* che *ToneControl*. In questo caso, l'oggetto può essere usato per controllare il volume che la generazione dei toni.
I Control sono resi disponibili al Player solo dopo che è stato riconosciuto il "content-type" del media.

4.4 JMF

L'architettura Java Media Framework (JMF) nasce con l'intento di potere lavorare su contenuti multimediali utilizzando il linguaggio Java. Permette di operare su media come audio e video clips, sequenze MIDI, e animazioni. Con JMF ad esempio è possibile :

- Eseguire vari file multimediali in applicazioni Java o Java applet. I formati supportati includono AU, AVI, GSM, MPEG, QuickTime e WAV.
- Eseguire streaming da internet.
- Catturare audio e video con microfono e video camera e memorizzarlo scegliendo tra i formati supportati.
- Processare contenuti multimediali time-based e cambiare il content-type format.
- Trasmettere audio e video in realtime in internet (RTP).
- Fare live broadcast di programmi radio o televisivi
- …

JMF fa uso di numerose classi, tra cui:

- Data Source
- Capture Device
- Player

- Processor
- DataSink
- MediaLocator
- Format
- Manager

Vediamole nel dettaglio.

4.4.1 Data Source

In JMF un oggetto DataSource incapsula una sorgente audio, video, o una combinazione di entrambe. Un Datasource può essere un file o uno streaming da internet. Il vantaggio di avere un oggetto DataSource è che una volta determinata la locazione o il protocollo, tale oggetto incapsula sia la locazione, protocollo e software per la gestione del media.
Una volta creato, il DataSource può essere passato ad un Player per essere eseguito, o ad un Processor per essere processato, disinteressandosi da come è stato realizzato il DataSource e quale era la sua forma originale.
Un DataSource può essere classificato in base a come il trasferimento ha inizio:

- **Pull data source**: il client inizia il trasferimento dati e controlla il flusso dalla sorgente. Esempi di questo tipo sono il protocollo HTTP e FILE.
- **Push data source**: Il server inizia il trasferimento dati e controlla il flusso. Esempi di questo tipo possono essere broadcast e video on demand.

Diversi DataSource possono essere combinati tra loro, ad esempio se stiamo catturando da una video camera avremo 2 DataSource: audio e video. In questo caso possiamo combinarli per un controllo più semplice.

4.4.2 Capture Device

Un Capture Device rappresenta un dispositivo sorgente, come un microfono, una webcam o una video camera. Il media catturato può essere passato ad un Player per essere riprodotto o processato per essere convertito in un altro formato, o memorizzato per un uso futuro.
Anche un Capture Device può essere pull o push. Se è una sorgente di tipo pull è l'utente che controlla quando iniziare l'acquisizione, come ad

esempio il click fatto su una webcam per ottenere una foto. In contrapposizione una sorgente di tipo push invia continuamente uno stream, come ad esempio un microfono o una webcam.

4.4.3 Player

Un Player prende come input uno stream audio o video e lo riproduce. Un player ha sei stati, attraversi i quali prima viene impostato e può poi partire:

- **Unrealized**: In questo stato , l'oggetto Player non è stato istanziato. Non conosce nulla sul media.
- **Realizing**: Si ha il passaggio da stato unrealized a stato realizing quando si invoca il metodo realize(). In questo stato, il Player è in attesa di determinare le risorse richieste. Spesso in questo stato esegue il download delle risorse dalla rete.
- **Realized**: Transitando dallo stato realizing il Player arriva allo stato realized. In questo stato il player conosce le risorse di cui necessita e ha informazioni sul tipo di media che deve presentare. Può anche fornire componenti visuali e controlli, e vengono completate eventuali connessioni ad altri oggetti nel sistema.
- **Prefetching**: Quando viene invocato il metodo prefetch(), il player passa dallo stato realized a quello di prefetching. Un Player nello stato di prefetching si prepara a presentare il media. In questa fase il Player precarica il media, ottenendo l'uso esclusivo sulla risorsa ed esegue tutte le operazioni necessarie all'esecuzione.
- **Prefetched**: Stato in cui il Player ha finito di eseguire il prefetching del media ed è pronto a partire.
- **Started**: Il Player entra in questo stato quando chiamo il metodo start(). Il Player è pronto a presentare il media.

4.4.4 Processor

Un Processor è un tipo di Player. In JMF API, l'interfaccia Processor estende l'interfaccia Player. Quindi ha gli stessi controlli di un Player. Diversamente dal Player un Processor ha controllo su cosa sta processando.

In aggiunta alla visualizzazione di un data source, un Processor può in uscita inviare il media attraverso un DataSource, e può essere dato come input ad un altro Player o Processor, processato da un altro Processor, o convertito in un altro formato.
In aggiunta ai 6 stati del Player sopra discussi, un Processor ha altri 2 stati che raggiunge prima dello stato realizing, ma dopo lo stato unrealized:

- **Configuring:** Un Processor entra nello stato configuring dallo stato unrealized invocando il metodo configure(). Un Processor raggiunge lo stato configuring quando si connette al DataSource, fa il multiplexing dell'input stream e accede ad informazioni sul formato del media in input.
- **Configured:** Dallo stato configuring un Processor si sposta nello stato configured quando è collegato al DataSource e il formato del media è stato determinato.

Come un Player, un Processor transita nello stato realized quando è chiamato il metodo realize().

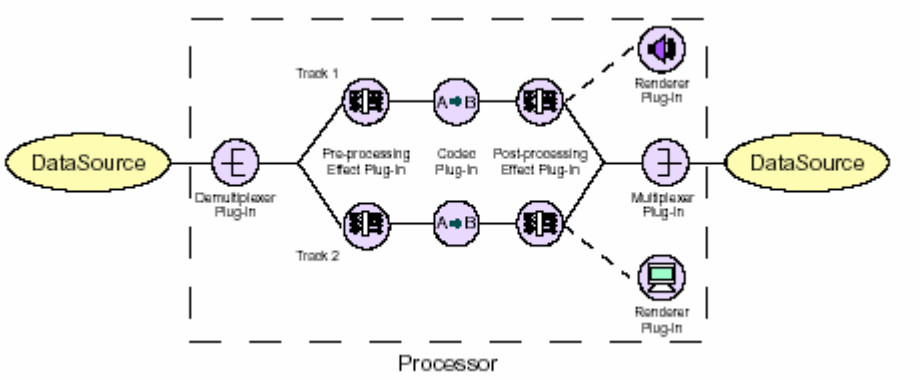


Figura 4.11 - Stadi di un Processor

I Processor sono creati nello stesso modo con cui sono creati i Player. Il Manager fornisce un metodo *createRealizedProcessor* per creare il Processor. E' possibile utilizzare un *ProcessorModel* per creare un Modello di Processor e tramite il metodo *Manager.createRealizedProcessor*, che accetta un ProcessorModel come parametro e che aderisce al ProcessorModel. Il Processor appena creato si trova nello stato Realized. Per programmare un Processor dobbiamo specificare il formato del media.

modello = new ProcessorModel(outputFormats, outputContainer);
processor = Manager.createRealizedProcessor(modello);

4.4.5 Data Sink

DataSink è un'interfaccia per oggetti che prendono un media da un DataSource e lo passano verso qualche destinazione. La destinazione è specificata con un MediaLocator.
mySink = Manager.createDataSink(source, sinkLocation);
Un esempio di DataSink è un oggetto che memorizza il media su un file.

4.4.6 Media Locator

MediaLocator descrive la locazione di un media. MediaLocator è in stretta relazione con le URL. URLs possono essere ottenute da un MediaLocator e un MediaLocator può essere costruito da un URL. Diversamente da un URL, un MediaLocator può essere instanziato senza un URLStreamHandler installato sul sistema.

4.4.7 Format

Un oggetto Format astrae un formato di un media specifico. Non ha parametri per una codifica specifica o informazioni sulla durata globale della presentazione. Descrive il nome di un encoding e tipo di dati che il formato richiede. Sottoclassi di Format includono AudioFormat e VideoFormat. Andando ad esplorare la gerarchia di classi troviamo che VideoFormat contiene 6 sottoclassi dirette:

- *H262Format*
- *H263Format*
- *IndexedColorFormat*
- *JPEGFormat*
- *RGBFormat*
- *YUVFormat*

4.4.8 Manager

Manager è il punto di accesso per ottenere le risorse dal sistema come Players, DataSources, Processors, DataSinks, il sistema TimeBase, le

utility cloneable e merging DataSources. Ad esempio con Manager posso creare un Player da un DataSource. JMF offre quattro manager:

- **Manager**: Necessario per creare Players, Processors, DataSources e DataSinks. Per esempio per fare il rendering di un DataSource possiamo usare Manager per creare un Player adatto.
- **PackageManager**: Questo manager mantiene un registro di packages che contengono classi JMF, come Players, Processors, DataSource e DataSinks personalizzati.
- **CaptureDeviceManager**: Questo manager mantiene un registro dei dispositivi di acquisizione.
- **PlugInManager**: Questo manager mantiene un registro di plug-in JMF per il processing.

4.4.9 JMF Registry

JMF permette di utilizzare tutte le periferiche di acquisizioni installate sul sistema, ma affinché sia possibile utilizzarle occorre che siano impostate correttamente anche per l'ambiente JMF. JMFRegistry è un'applicazione che registra nuovi DataSource, MediaHandlers, PlugIns e dispositivi di acquisizione.
E' necessario chiamare questa utility solo una volta per registrare tutti i dispositivi di acquisizione presenti sul sistema.
Per ogni dispositivo trovato poi vengono elencati tutti i formati che supporta.

4.5 Confronto tra MMAPI e JMF

La progettazione delle MMAPI è stata influenzata dal JMF, e hanno molte cose in comune. Entrambe sono API che lavorano ad alto livello con contenuti multimediali time-based ed hanno molte somiglianze. Entrambi condividono il concetto di Player e DataSource. Dal momento che le API sono così simili programmi basati su JMF possono essere portati su MMAPI.
Rispetto alle MMAPI, JMF è stato pensato per essere usato in ambiente server o client, dove ci sono risorse di CPU e memoria sufficienti ad eseguire agevolmente applicazioni Java. JMF 2.1 ha anche un numero di

funzionalità che non sono presenti nelle MMAPI, come la possibilità di transcodifica di uno stream di dati.
JMF ha un numero di caratteristiche che non lo rendono adatto per un piccolo dispositivo. Ad esempio, con le restrizioni CLDC, il dispositivo non supporta operazioni in virgola mobile, AWT, e un certo numero di oggetti Java presenti nelle specifiche del JMF. Anche la dimensione delle API è un problema: il JMF non può essere portato a dimensioni di 20KB come richiesto dai dispositivi portatili. Per questo motivo in questi dispositivi è stato necessario introdurre nuove API.
Le MMAPI sono rivolte a client con risorse limitate, dove interagiscono con l'hardware e software nativo. Le MMAPI non permettono ingrandimento di immagini e non sono progettate per essere usate in applicazioni 2D o 3D, né dovrebbero essere usate come API per giochi, cioè in applicazioni dove è presente un alto grado di interazione. C'è poco beneficio ad utilizzare ad usare le MMAPI laddove è possibile utilizzare JMF.
Il JMF rispetto alle MMAPI supporta tipi di media addizionali e permette di eseguire azioni di riproduzione e processing. Il JMF supporta la cattura e la memorizzazione dei dati, l'utilizzo e lo sviluppo di plugin per codifica e decodifica.
Per la gestione degli eventi in JMF con l'interfaccia *javax.media.ControllerListener* abbiamo un ampio insieme di funzionalità. Possiamo catturare eventi come *RealizeCompleteEvent, PrefetchCompleteEvent, StartEvent, StopEvent*.
Con le MMAPI, possiamo utilizzare l'interfaccia *javax.microedition.media.PlayerListener* ma è molto più limitante: tra gli eventi che vengono rilevati troviamo *END_OF_MEDIA*: fine del media, *CLOSE*: il player viene chiuso, *STARTED*: il player viene avviato, *STOPPED*: il player è stato fermato dal metodo *stop()*. Non viene catturato il raggiungimento dello stato *PREFETCHED* del player.

4.6 Conclusione

In questo capitolo abbiamo visto brevemente le varie distribuzioni Java, e ci siamo soffermati sull'architettura J2ME. Abbiamo visto quali sono i requisiti e le caratteristiche che contraddistinguono questa distribuzione.

In particolare abbiamo visto che cos'è, come funziona e come deve essere realizzata una MIDlet, facendo riferimento anche alla struttura sottostante. Per lavorare con contenuti multimediali sono stati trattati le MMAPI e JMF, andando ad analizzare il funzionamento, le differenze e le motivazioni che hanno spinto a realizzare questi due gruppi di API.

CAPITOLO 5

Implementazione del progetto

In questo capitolo andremo ad analizzare come le scelte architetturali sono state implementate andando ad utilizzare un linguaggio di programmazione come Java.
Analizzeremo le due entità in gioco: server e client, spiegando quali componenti e quali protocolli di comunicazione sono stati utilizzati.

5.1 Scelte implementative

Nell'implementazione del progetto è stato utilizzato il linguaggio Java. Lato server è stata utilizzata la J2SE (Java 2 Standard Edition) mentre sul client, a causa delle limitazioni di risorse dei dispositivi mobili è stata utilizzata la J2ME (Java 2 Micro Edition). Come abbiamo visto nel precedente capitolo lavorando con contenuti multimediali abbiamo utilizzato strumenti diversi.
Lato server è stato possibile utilizzare il JMF (Java Media Framework), sul dispositivo mobile abbiamo utilizzato MMAPI (Mobile Media API).
La configurazione mobile supporta CLDC 1.1 e MIDP 2.0.
Per la fase di comunicazione è stato utilizzato il protocollo HTTP, disponibile per tutte le piattaforme.

5.1.1 Implementazione Server

Il server Web è nato come programma specializzato nel trasferimento di risorse attraverso il protocollo HTTP. Con l'evoluzione del Web, il server ha acquisito tutta una serie di funzionalità aggiuntive ed è ora in grado di eseguire elaborazioni e invocare programmi esterni, grazie a tecnologie come CGI, PHP, JavaScript lato server, Java Servlet e JSP, che lo hanno trasformato in vero e proprio ambiente per l'esecuzione di applicazioni.
La nostra implementazione si appoggia su un server web in grado di eseguire Java Servlet.
Il server è stato realizzato quindi come servlet che utilizza classi Java e JMF.

Il server una volta attivato rimane in ascolto, in attesa di richieste. Tramite il protocollo HTTP viene eseguita la richiesta e vengono passati i parametri.
Se ci sono la richiesta è formulata correttamente e ci sono i presupposti, inizia l'elaborazione e viene inviata una risposta positiva al client.
La fase di comunicazione è realizzata all'interno della servlet, la fase di acquisizione ed elaborazione viene realizzata da un thread.

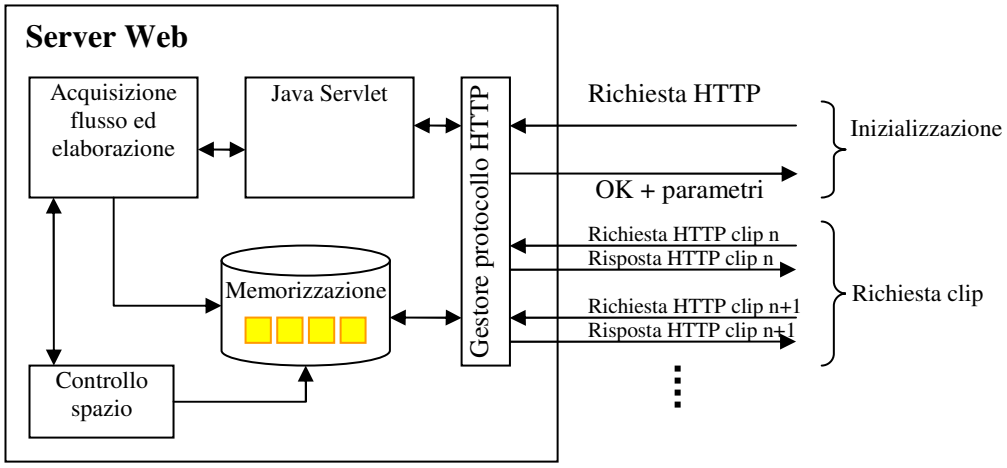


Figura 5.1 - Server

Affinché la servlet possa ricevere parametri e venire eseguita è necessario ridefinire il metodo *doGet(HttpServletRequest request, HttpServletResponse response)*, questi parametri vengono utilizzati per ricevere ed inviare parametri alla MIDlet.
La servlet dopo la fase di inizializzazione lancia l'esecuzione del thread per la frammentazione del flusso e un thread per il controllo dello spazio su disco.

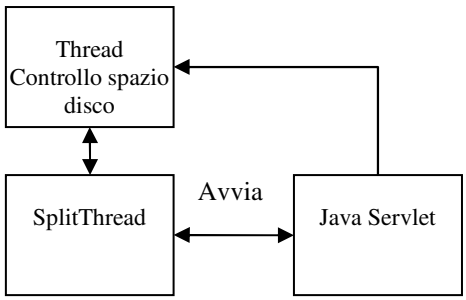


Figura 5.2 - Vengono lanciati 2 thread, per l' elaborazione del flusso e il controllo dello spazio di memorizzazione

Lo *SplitThread* deve:

- acquisire la risorsa;
- effettuare i collegamenti tra le varie componenti del sistema;
- iniziare la cattura dello streaming
- informare la servlet della sua esecuzione.

Il thread di controllo dello spazio del disco deve:

- controllare se il numero delle clip sta esaurendo lo spazio dedicato per la loro memorizzazione ed elimina le clip più vecchie in relazione alla produzione di quelle nuove.

Vediamo queste fasi nel dettaglio

5.1.1.1 Acquisizione della sorgente

Per il reperimento della sorgente di streaming è necessario chiedere all'oggetto di sistema *CaptureDeviceManager* quali dispositivi sono in grado di supportare un certo formato di acquisizione: *CaptureDeviceManager.getDeviceList(aformat)* con *aformat* il formato richiesto (es. *AudioFormat("linear",11025,16,1)).*

Il sistema restituisce una lista di periferiche compatibili con le specifiche e si utilizza la più opportuna. Nel nostro caso viene restituito un riferimento al microfono.

5.1.1.2 Setup dei componenti

Lo SplitThread ha necessità si eseguire una serie di operazioni di collegamento tra le varie parti prima di eseguire la cattura dello streaming.

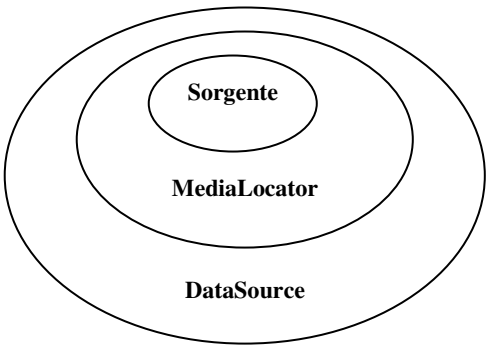


Figura 5.3 - Utilizzo della sorgente di streaming

Una volta ottenuta la sorgente di streaming, si ottiene il *MediaLocato*r di questo oggetto chiamando il metodo *getLocator()* su di esso: *audioMediaLocator = captureAudioDevice.getLocator()*.
Dal *MediaLocator* con l'ausilio del *Manager* si ottiene un *Datasource: audioDataSource = Manager.createDataSource(audioMediaLocator)*.
Quest'approccio permette di lavorare sempre ad un livello di astrazione maggiore, disinteressandosi completamente dei livelli sottostanti.
Viene a questo punto creato un modello per il *Processor* dal *Datasource* ottenuto, specificando il formato di uscita, e specificando il tipo di dati in uscita (Es. *FileTypeDescriptor.WAVE*): *processorModel = new ProcessorModel(audioDataSource, audioFormat, outputType)*.
Dal *ProcessorModel* è possibile ottenere un *Processor* in questo modo: *processor = Manager.createRealizedProcessor(processorModel)*.
Dal un *Processor* è possibile ottenere il flusso di uscita con il metodo *getDataOutput()* che restituisce un *DataSource*: *source = processor.getDataOutput()*.

5.1.1.3 Cattura dello streaming

Come abbiamo visto nel capitolo 4 il *Processor* permette di elaborare il flusso e di darlo in uscita ad esempio ad un oggetto che lo scrive su file. A questo scopo viene creato un oggetto *MediaLocator* con il percorso del file da creare: *dest = new MediaLocator("file://" + file)* dove *file* è una stringa che rappresenta il percorso del file. Il *MediaLocator* di destinazione viene passato ad un *DataSink*, insieme al flusso di uscita del *Processor: filewriter = Manager.createDataSink(source, dest)*.
Il *DataSink* scrive il file: *filewriter.open(); filewriter.start()* quando il *Processor* viene avviato: *processor.start()*.
Il *Processor* dopo un tempo prestabilito viene fermato *processor.stop()* e chiuso *processor.close()*. Il *DataSink* viene fermato e chiuso *filewriter.stop()*; *filewriter.close()* e il file viene scritto e risulta subito pronto per essere scaricato. La cartella di memorizzazione dei file è nota e raggiungibile via HTTP dal client.
Utilizzando il *ProcessorModel* è possibile creare nuovi Processor per continuare a salvare le porzioni di stream successive in altrettanti file per tutta la durata della presentazione.

5.1.1.4 Comunicazione con la Servlet

La servlet avvia il Thread di elaborazione, questi inizia a catturare e a spezzare il flusso. La servlet attende una risposta dal Thread che confermi il corretto avvio del sistema. Quando il Thread ha realizzato la prima clip, informa la servlet e a sua volta la servlet comunica al client che può scaricare.

5.1.1.5 Controllo dello spazio su disco

La servlet avvia anche un altro thread: il *CheckHDThread* che controlla che il numero delle clip memorizzate su disco non superi una certa soglia.
Ottiene la lista dei file quando si raggiunge il limite cancella quelli più vecchi. Tramite il metodo *setMaxPiece(int)* viene impostato il massimo numero di clip memorizzabili, modificabile in ogni momento.
Un'altra soluzione poteva essere quella di eseguire una numerazione modulo N, cioè ricominciare la numerazione dopo l'N-esima clip, andando a sovrascrivere le prime. Come soluzione è più semplice, ma più vincolante e meno configurabile: se un client fa una richiesta di una clip con una certa differita potrebbe trovarla sovrascritta.

5.1.2 Implementazione Client

Lato client, la MIDlet è composta da una serie di metodi che realizzano il controllo necessario per la fase di inizializzazione della comunicazione.
Sono impiegati due thread che svolgono rispettivamente i ruoli di Produttore e Consumatore delle clip.
Quando il Consumatore non riesce ad eseguire la clip successiva perché trova il buffer vuoto si passa è il controllo di sessione che decide da dove ripartire. Buffer vuoto significa problemi di collegamento e pertanto al ripristino della connessione si deciderà da quale clip fare ripartire il Produttore. Tenendo conto del periodo di inattività e si può decidere se riprendere da dove ci si era interrotti oppure da un altro punto.
La nostra scelta per interruzioni di riproduzione fino a 10 secondi è stata di fare ripartire dall'ultima clip non ascoltata, per periodi superiori si saltano le clip non ascoltate (es. per clip di 5 secondi con un ritardo tra i 10 e i 15 secondi salto 2 clip). In questo modo manteniamo la sessione controllando i due thread.

Il thread produttore crea tanti player con il vincolo di capacità del buffer. Viene utilizzato un *Vector* per memorizzare i player. Quando il *Vector* raggiunge la massima capacità il Produttore attende la liberazione di un posto. Un player viene creato passandogli l'indirizzo della clip da riprodurre: *player = Manager.createPlayer(url).*

I player vengono creati e portati nello stato *PREFETCHED*: *player.prefetch()*. Si lavora in maniera sequenziale.

Una volta raggiunto lo stato prefetched può essere inserito nel buffer *buf.insertPlayer()*. Il Consumatore al termine della riproduzione di una clip chiama il metodo *getNextPlayer()* sul buffer ed ottiene il player da eseguire: *buf.getNextPlayer()* e lo manda in riproduzione con *player.start()*.

Il Consumatore implementa l'interfaccia *PlayerListener* ed il termine di una clip lancia un evento di tipo *END_OF_MEDIA*: si deve avviare un altro player. Se viene restituito *null* significa che il produttore non è stato un grado di scaricare la clip successiva per un problema di collegamento e pertanto il Produttore si blocca in attesa di essere riattivato dal Controllo di Sessione. Ci saranno metodi per conoscere lo stato del buffer: il Produttore prima deve controllare che ci sia spazio sul buffer con il metodo *buf.isFull().*

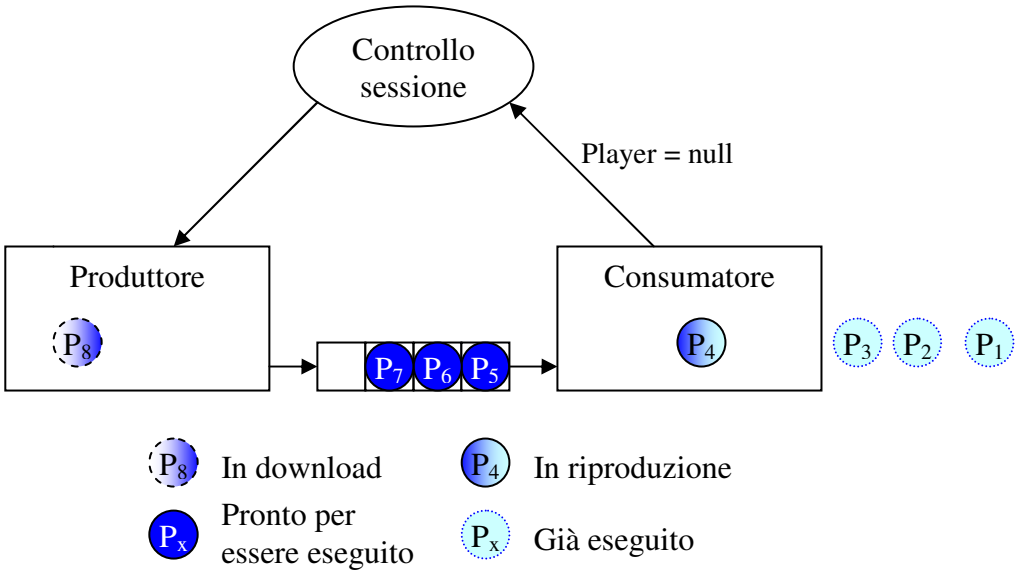


Figura 5.4 - La MIDlet utilizza due Thread: un produttore e un consumatore

La parte che ha richiesto una maggiore attenzione è stata quella della sincronizzazione.
La MIDlet durante la riproduzione presenta sul display una serie di informazioni utili per valutare la qualità del servizio come i tempi tra l'*END_OF_MEDIA* e la *start()* della clip successiva.
Per ottenere il display del dispositivo è sufficiente eseguire: *Display.getDisplay(this).setCurrent(screen)*, dove *screen* è un *Form*, per scriverci basta usare *screen.append("Text")*.

5.2 Comunicazione

Come abbiamo già diverse volte anticipato, la comunicazione avviene utilizzando il protocollo HTTP, sia per la fase di inizializzazione che per la fase di download delle clip realizzate. Vediamo meglio i protocolli di comunicazione.

5.2.1 Inizializzazione

Il server web è in attese di richieste su una certa porta e la servlet viene avviata specificando l'host, la porta e la Java Servlet.

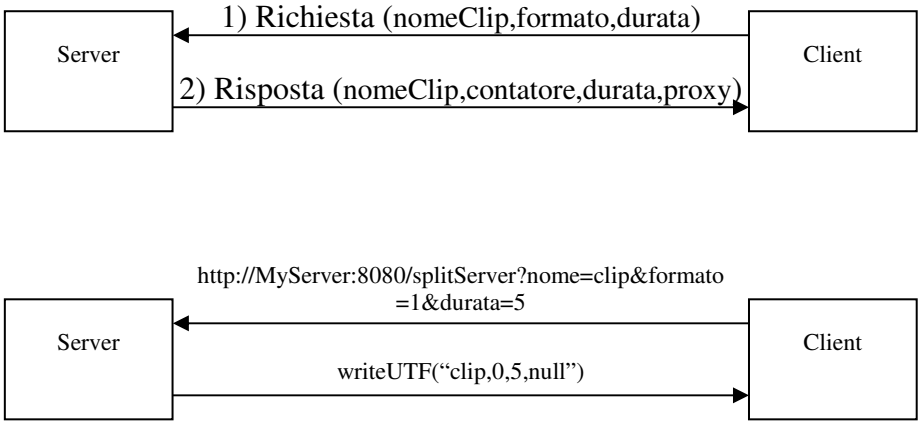


Figura 5.5 - Protocollo di inizializzazione

Abbiamo ridefinito all'interno della Servlet il metodo *doGet(HttpServletRequest request, HttpServletResponse response)*.

La richiesta da parte delle MIDlet viene fatta utilizzando l' *HttpConnection* passando come parametro un URL che ha questa forma: *http://<HostServerWeb>:<porta>/<NomeServlet>?<parametro1>&<parametro2>&…<parametroN>*.
Verrà specificata che la richiesta è di tipo *GET* e varranno passate una serie di informazioni che qualificano il client, come lo User-Agent che nel nostro caso è *Profile/MIDP-2.0 Configuration/CLDC-1.1*, la lingua, il formato dei dati utilizzato.
Vediamo i vari passi come da figura 5.5:

1. Il client esegue la richiesta al server con: *HttpConnection conn = (HttpConnection) Connector.open(url, Connector.READ_WRITE)* dove url è la stringa completa compresi i parametri (es. *http://MyServer:8080/splitServer?nome=clip&formato=1&durata=5*).
2. Il server per leggere i parametri utilizza *request.getParameter(<parametro>)* (es. *request.getParameter("nome")*) dove *request* è un *HttpServletRequest*, parametro della *doGet( )*.
3. Il server quando si può procedere al download perché le clip sono state già preparate invia l'ok con *response.setStatus(HttpServletResponse.SC_OK)* altrimenti con *response.setStatus(response.SC_BAD_REQUEST)* dà una risposta negativa.
4. Il client legge l'esito con *conn.getResponseCode()*. Se l'esito è *HttpConnection.HTTP_OK* posso continuare.
5. Il server invia i parametri come il nome della clip, il contatore, la durata di ciascuna clip in secondi e il proxy se esiste: *dos.writeUTF(<nomeClip>,<contatore>,<durata>,<proxy>)*
   come ad esempio *dos.writeUTF("clip,0,null")* con *dos DataOutputStream* creato da un *ByteArrayOutputStream*.
6. Il client legge la stringa con *dis.readUTF()* dove *DataInputStream dis = new DataInputStream(conn.openInputStream())* e interpreta i parametri.

La MIDlet a questo punto sa da dove può cominciare a scaricare le clip e la durata.
Nella risposta vengo nuovamente riportati nome base delle clip, il contatore da cui partire, e la durata perché se c'è già una precedente elaborazione avviata questi parametri sono già stati impostati e il client deve prenderne atto. La durata serve per il dimensionamento del buffer.

5.2.2 Download delle singole clip

Quando il server è avviato il client dopo la fase di inizializzazione sa come reperire le singole clip, se dal sever o dal proxy, la loro durata. Il download delle clip è affidato al produttore ed è lui in base alla disponibilità sul server e alla capacità locale stabilire quante ne devono scaricare. In ogni caso il download è sequenziale, cioè si attende di avere scaricato una clip audio prima di iniziare il download della successiva. Non ci sarebbero vantaggi ad utilizzare un download parallelo quando la presentazione e sequenziale: anche se complessivamente potrebbe aumentare la banda utilizzata sicuramente allungherebbe il tempo di download della clip da eseguire nell'immediatezza e a fronte di errori nella comunicazione occorrerebbe anziché riscaricare una clip riscaricare da capo *n* clip, con una notevole peggioramento.
Il produttore crea un player in questo modo:
player = Manager.createPlayer(url) con *url* della forma *http://myServer:port/path/clipN.wav*.
Ancora una volta notiamo che il protocollo usato è l'HTTP. Quando sul player viene fatta la chiamata *player.prefetch()* il player acquisisce le risorse dalla rete e raggiunge lo stato *PREFETCHED* e può essere inserito nel buffer.
Nella fase di inizializzazione sappiamo il nome e i numero della clip, le successive utilizzano lo stesso nome con la numerazione progressiva.
Se la comunicazione viene interrotta per un certo lasso di tempo, il Controllo di Sessione deciderà da dove il produttore deve continuare a riprendere a scaricare.

5.3 Conclusione

In questo capitolo, partendo dalle scelte architetturali esposte nel capitolo 3, abbiamo cercato di esporre come sono stati implementate la varie parti del sistema.

Lato server, abbiamo discusso dell'implementazione della servlet e delle classi accessorie. Lato client, la realizzazione della MIDlet che utilizza due thread, un Produttore e un Consumatore. Abbiamo parlato della comunicazione sia tra client e server, che tra le varie componenti dell'applicazione, con un riguardo al problema della sincronizzazione. Lato server abbiamo utilizzato Java Standard Edition, con alcune classi specifiche per la realizzazione di Java Servlet, e per la componente multimediale abbiamo utilizzato il Java Media Framework (JMF).

Lato client la MIDlet è stata implementata utilizzando le classi secondo le specifiche J2ME e utilizzando per la parte multimediale le librerie Mobile Media API (MMAPI).

CAPITOLO 6

Ambiente di sviluppo e test sul campo

In questo capitolo parleremo prima degli ambienti di sviluppo utilizzati, per la fase di test di sistema in fase di emulazione, poi vedremo gli strumenti utilizzati per testare sul campo il risultato del nostro lavoro. Vedremo l'ambiente di sviluppo Eclipse, e i suoi plugin, che permettono di lavorare in un ambiente integrato nella realizzazione di una MIDlet Suite. Parleremo del server web Apache Jakarta Tomcat, che rende possibile l'utilizzo di servlet e quindi l'elaborazione dello stream multimediale che deve essere presentato al dispositivo portatile. Ci sposteremo poi sul campo, con dispositivi reali, parlando prima delle tecnologie utilizzate e testando il sistema, analizzando i risultati ottenuti.

6.1 Eclipse

Eclipse è un ambiente di sviluppo IDE (Integrated Development Environments) per lo sviluppo di software indipendente dalla piattaforma. Inizialmente fu sviluppato da IBM, ma attualmente è sviluppato dall'Eclipse Foundation, un consorzio no-profit che comprende numerose industrie, tra cui Borland, IBM, HP, per citarne alcune [Eclipse].
L'architettura di Eclipse si basa su Rich Client Platform (RCP), che comprende i seguenti componenti:

- Core platform, boot Eclipse, run plugin;
- OSGi, un ambiente standard di lavoro;
- SWT, un sistema di sviluppo portabile;
- JFace, strumenti per l'utilizzo di file, elaborazione di testi;
- The Eclipse Workbench, viste, editor, prospettive, wizard.

Eclipse impiega plugin per fornire funzionalità aggiuntive.
Quelli da noi utilizzati sono il plugin eclipseME e il plugin per Tomcat.

6.1.1 Plugin eclipseME

Il plugin per eclipse eclipseME [EMEP] permette di utilizzare Eclipse per la realizzazione di applicazioni J2ME MIDlet. Svolge l'attività di collegare il Wireless Toolkit all'ambiente di sviluppo Eclipse. Una volta

installato va configurato associando all'ambiente J2ME il Wireless Toolkit.

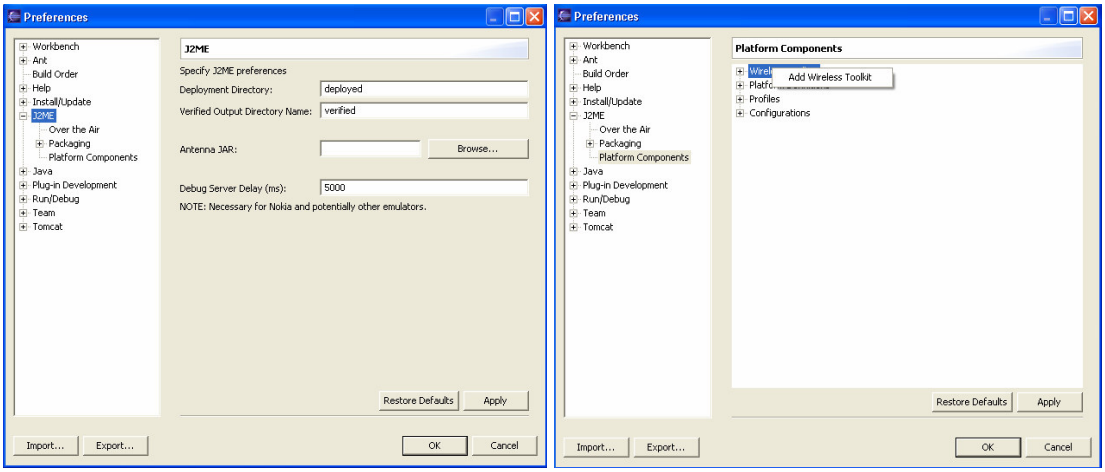


Figura 6.1 - Aggiunta del Wireless Toolkit a eclipseME

6.1.2 Plugin Sysdeo Eclipse Tomcat Launcher

Un altro plugin che abbiamo utilizzato è il plugin Sysdeo Eclipse Tomcat Launcher [SYS04], che una volta installato permette di:

- Far partire, fermare e far ripartire Tomcat;
- Registrare i processi Tomcat nel debugger di Eclipse;
- Creare un progetto WAR (Web Archive) (il wizard può aggiornare il file server.xml);
- Aggiunge progetti Java al classpath di Tomcat;
- Imposta i parametri JVM di Tomcat, classpath e bootclasspath;
- Esporta un progetto Tomcat in un file WAR;
- Permette di scegliere i file di configurazione di Tomcat.

6.2 Wireless Toolkit

Il Wireless Toolkit [WT04] è un'emulatore di ambiente wireless che permette la compilazione e il debug di applicazioni J2ME.
Con il Wireless Toolkit è stato possibile eseguire la fase di progetto e di emulare test prima di procedere all'utilizzo di dispositivi concreti e con risultati reali.

Il Wireless Toolkit fornisce un ambiente di esecuzione e gestione delle applicazioni, supporta lo sviluppo di applicazioni J2ME che possono venire eseguite su dispositivi con un Mobile Information Device Profile (MIDP), come PDA e cellulari e lo sviluppo di applicazioni Wireless Messaging API (WMA) e Mobile Media API (MMAPI).
La KToolbar, inclusa nel Wireless Toolkit, è un'interfaccia grafica (GUI) per la compilazione e l'esecuzione di applicazioni MIDlet.
Un ambiente IDE compatibile, come ad esempio Eclipse, semplifica molto il lavoro di sviluppo perché dal suo interno è possibile editare, compilare ed eseguire un'applicazione MIDlet con il Wireless Toolkit.
Il Wireless Toolkit fornisce un'implementazione del MIDP insieme ad un emulatore che può essere utilizzato per la visualizzazione di un risultato su un display. Non è un comunque un ambiente completo, perché non fornisce un editor integrato che permette di realizzare, vedere e modificare il codice sorgente. Di conseguenza abbiamo utilizzato Eclipse per permettere un utilizzo più comodo e immediato di questo tool di sviluppo.

6.2.1 Esecuzione di una MIDlet

Il Wireless Toolkit è configurabile e personalizzabile. Per un buon utilizzo dell'emulatore è necessario settare le dimensioni dello spazio di storage e dello spazio dedicato all'heap. Questo per specificare la memoria disponibile per le applicazioni che dovranno essere eseguite su un dispositivo che supporta le MMAPI.

Heap Size	MIDlet Type
64K	Single Tone
64K	Tone Sequence
64K	Bouncing Ball demo, includes WAV version, Tone Sequence version and a version without background music
64K	Snake demo
128K	MPEG-1 video playback from file
256K	MIDI playback simple player midlet playing file (`test-midi.mid`) or HTTP (`test-midi.mid`)
256K	WAV/PCM audio playback (simple player midlet playing file, `bong.wav`, or http, `javaone.wav`)
256K	WAV/PCM audio record (simple player midlet, capture mode)
512K	Video playback from compressed local source (`.jar`)

Tabella 6.1 - Dimensione minima dell'heap per una MIDlet

Il valore dell'heap che di default è 4000KB, i valori ammessi vanno da 32 KB a 64000 KB. Il valore minimo dell'heap dipende dal tipo di MIDlet che si sta eseguendo. Per rendersi conto delle dimensioni richieste da varie applicazioni si consulti la tabella 6.1.
E' possibile anche modificare i parametri di qualità e velocità della trasmissione radio.
La MIDlet avviata presenterà un'interfaccia grafica simile a quella della figura 6.2, dove possiamo interagire con i pulsanti per comandarla.

Figura 6.2 - Emulatore Wireless Toolkit

Quando una MIDlet viene caricata, il gestore delle applicazioni mostra la lista delle midlet che sono presenti e permette di selezionarne una ed iniziare la sua esecuzione. Pensando alla sicurezza potrebbe essere pericoloso per un dispositivo lanciare una MIDlet automaticamente.
L'utente vedrà la lista delle MIDlet da eseguire così come è stato riportato nel file *manifest.mf* seguendo gli attributi (MIDlet-1, MIDlet-2, … ,

MIDlet-n). E possibile associare un'icona a una MIDlet specificandolo nel file *manifest.mf*.
L'installazione sul dispositivo è possibile utilizzando i file JAR e JAD.

6.3 Apache Jakarta Tomcat Servlet/JSP Container

Tomcat è scritto in Java e fa parte del progetto Apache [Tomcat05]. È un contenitore di servlet e Java Server Pages (JSP), cioè un server Web orientato alla gestione di servlet e JSP, tuttavia può funzionare anche come un server Web tradizionale e quindi è in grado di fornire pagine HTML statiche e interfacciarsi con programmi CGI. La versione attuale implementa le specifiche servlet versione 2.4 e JSP versione 2.2 del Java Community Process, e include alcune caratteristiche aggiuntive che lo rendono un'efficiente piattaforma per lo sviluppo e l'esecuzione di applicazioni e servizi Web [Tomcat05]. I servizi che non riguardano servlet e JSP sono implementati anch'essi tramite servlet specifiche.
Il Default Servlet è il modulo che fornisce le risorse statiche e il listing delle directory, se abilitato. Con risorsa statica si intende tutto ciò che non è servlet o JSP, come pagine HTML, immagini, ecc. È possibile modificare o addirittura sostituire il Default Servlet con una propria implementazione, per conferire al server il comportamento desiderato.
CGI definisce un modo per far interagire un server Web con programmi esterni che generano contenuti, che vengono detti programmi CGI oppure script CGI. Il supporto a CGI può venir aggiunto a Tomcat quando esso viene utilizzato come server HTTP tradizionale, tipicamente solo durante la fase di sviluppo, e presenta qualche limitazione rispetto ad altre implementazioni, come Apache, che sono invece progettate appositamente per gestire l'interfaccia CGI.
Il supporto a CGI viene implementato utilizzando il modulo CGIServlet e di default è disabilitato.

6.3.1 Java Servlet

Una servlet è un programma scritto in Java che viene eseguito all'interno di un server Web capace di gestirlo. Il server web perciò viene detto servlet container [Servlet02]. Una servlet riceve e risponde alle richieste provenienti da client Web, effettuate di solito attraverso il protocollo

HTTP. L'implementazione standard delle servlet è costituita dai due package *javax.servlet* e *javax.servlet.http* e non è inclusa nell'ambiente di sviluppo Java 2 SDK Standard Edition, ma è presente solo nell'Enterprise Edition. Anche il server Web Apache Tomcat ne contiene un'implementazione.

L'interfaccia *javax.servlet.Servlet* definisce i metodi per gestire quello che viene detto ciclo di vita della servlet:

- la servlet viene creata, poi viene inizializzata tramite il metodo init;
- ogni chiamata proveniente da un client viene gestita attraverso il metodo *service*, i cui due parametri di tipo *ServletRequest* e *ServletResponse* permettono di accedere al contenuto della richiesta e di restituire una risposta dopo averla costruita;
- la servlet cessa di fornire il servizio e viene distrutta attraverso il metodo destroy.

La classe astratta *javax.servlet.GenericServlet* è una implementazione di questa interfaccia e definisce una servlet generica indipendente dal protocollo. La sottoclasse *javax.servlet.http.HTTPServlet* fornisce una classe astratta che deve essere estesa per creare una servlet HTTP adatta per un sito Web. Normalmente lo sviluppatore ridefinisce almeno uno tra i seguenti metodi:

• *doGet* – se la servlet gestisce richieste HTTP di tipo GET;
• *doPost* – per richieste HTTP di tipo POST;
• *doPut* – per richieste HTTP di tipo PUT;
• *doDelete* – per richieste HTTP di tipo DELETE;
• *init* e *destroy* – per riservare e rilasciare le eventuali risorse esterne utilizzate durante il ciclo di vita della servlet;
• *getServletInfo* – che la servlet utilizza per fornire informazioni su di sé, come l'autore, la versione e il copyright.

Nella nostra implementazione abbiamo ridefinito il metodo *doGet( )*.

I parametri di tipo *HttpServletRequest* e *HttpServletResponse*, sottoclassi dei precedenti *ServletRequest* e *ServletResponse*, permettono alla servlet di accedere alla richiesta HTTP e di restituire la risposta. Normalmente non occorre ridefinire il metodo *service*: esso non fa altro che smistare ogni richiesta HTTP standard il base al tipo, inviandola al metodo

preposto a gestirla, di solito uno dei primi quattro della lista precedente. Di solito una servlet viene eseguita su un server multithread. Per questo motivo ogni servlet deve gestire in maniera opportuna le richieste concorrenti e deve sincronizzare l'accesso a risorse condivise, che includono dati in memoria, come istanze o variabili di classe, ed oggetti esterni come file, connessioni a basi di dati o alla rete.

6.3.1.1 Il file *web.xml*

Come accennato in precedenza il file /WEB-INF/*web.xml* contiene il Web Application Deployment Descriptor per l'applicazione. Come indica l'estensione, questo è un documento XML, e definisce ogni cosa riguardante l'applicazione che un server deve conoscere (ad eccezione del *context path*, che è assegnato dall'amministratore di sistema quando l'applicazione viene sviluppata).
La sintassi completa e la semantica sono riportate nelle specifiche delle Servlet API [Servlet02].
Nel file web.xml viene specificato tra le altre cose il nome da attribuirsi alla servlet, la classe con il main associata, il periodo di mantenimento della sessione.

6.4 IBM WebSphere Everyplace Micro Environment

IBM WebSphere Everyplace Micro Environment (WEME) Connected Limited Device Configuration (CLDC)/MIDP for Windows Mobile (TM) 2003 fa parte del pacchetto software IBM Workplace Client Technology, Micro Edition [WEME05].
Il cuore di WebSphere Everyplace Micro Environment (WEME) è J9 VM, implementazione IBM delle specifiche della Java Virtual Machine, versione 1.3. La J9 VM con le Java Class Libraries (JCL) costituiscono l'ambiente di esecuzione J9. L'ambiente runtime J9 è conforme alle specifiche della piattaforma Java 2 Micro Edition (J2ME) e include la Connected Limited Device Configuration (CLDC) e Connected Device Configuration (CDC). La versione supportata delle MIDP è la 2.0 e CLDC 1.1. Le classi della J9 VM sono implementazioni di J2ME Mobile Information Device Profile Next Generation (JSR-118), basate sulle specifiche MIDP (JSR-37).

6.5 Test sul campo

In questo paragrafo, utilizzando gli strumenti sopra descritti riportiamo la descrizione di quello che abbiamo fatto in fase di test, caricando, dopo una prima fase di test sull'emulatore del Wireless Toolkit, la nostra applicazione su dispositivi concreti.

6.5.1 Obiettivi

Le prove sperimentali hanno lo scopo di testare quello che è stato sviluppato e testato su un ambiente emulato, con il Wireless Toolkit, in un ambiente reale. Questo per analizzare i risultati di test sul campo per scoprire eventuali problematiche e discrepanze con i quelli ottenuti con l'emulatore e per dare valore maggiore ai nostri sforzi provando l'applicazione. Importante è stato vedere il sistema funzionare e prendere atto dei tempi di risposta, dei tempi di download delle parti, dei tempi di attesa tra una clip audio e l'altra.

6.5.2 Dispositivi utilizzati

I dispositivi che sono stati impiegati per i test sono:

- Computer portatile PIII 800, con 256MB di RAM, su cui è stato installato Apache Jakarta Tomcat 5.0.28, Java Standard Edition 1.4.2, JMF 2.1.1e

Figura 6.3 - IBM ThinkPad T21

- HP iPAQ h5500: Pocket PC con 128 MB di RAM e 48 MB di ROM. Sistema operativo installato Windows CE 4.2, con display 320x240 pixel a 64K colori TFT. Con processore Intel® PXA255. Dotato di connettività bluetooth, wireless 802.11 e infrarossi.

Figura 6.4 - HP iPAQ h5500

- D-Link DBT-900AP: Access point Bluetooth, collegato via LAN al server.

Figura 6.5 - D-Link DBT-900AP

Sul PDA è stata installata IBM WEME (WebSphere Everyplace Micro Environment) MIDP 2.0 per Windows Mobile 2003, che fornisce un'implementazione delle specifiche J2ME per CLDC (Connected Limited Device Configuration) versione 1.1 e MIDP (Mobile Information Device Profile) versione 2.0.

Figura 6.6 - Dispositivi utilizzati

E’ stata utilizzata esclusivamente la connettività Bluetooth tramite l’access point DBT-900AP.

6.5.3 Risultati

Per la fase di test si è deciso di utilizzare un formato audio wav (audio/x-wav). Senza compressione, in qualità a 16 bit, a 11025Hz di campionamento, mono.
La durata di ciascuna clip è stata scelta di 3, 5, 10 e 15 secondi. Per una dimensione di circa 64KB, 107KB, 215 KB e 323KB rispettivamente.
Una volta caricata la MIDlet in ambiente IBM WEME sul dispositivo si realizzata la connessione bluetooth con l’access-point.
Lanciata la MIDlet, le impostazioni di sicurezza indicano che il dispositivo necessita di collegarsi per una richiesta HTTP. Concesso il collegamento abbiamo verificato che i tempi di download di ogni singola clip, sono inferiori al tempo di riproduzione, abbiamo valutato il tempo necessario per il passaggio dalla fine di una clip all’inizio della successiva, l’overhead di trasmissione dovuto alla comunicazione, come riportato in tabella 6.1.

Durata (sec.)	Dim. (KB)	Tempo di download (ms)	Tempo di cambio contesto da clip n a clip n+1 (ms)									OverHead Max
			0/1	1/2	2/3	3/4	4/5	5/6	6/7	7/8	8/9	
3	64	1861	38	33	35	36	34	36	37	39	44	30-35%
5	107	3250	40	40	36	39	36	38	41	41	42	30-35%
10	215	5975	41	38	39	38	41	42	44	44	44	30-35%
15	323	8446	48	39	40	40	40	45	47	47	46	30-35%

Tabella 6.1 - Risultati sperimentali per formato audio

Il ritardo tra la fine di una clip e l’inizio della riproduzione della successiva mediamente è sui 40 ms, e l’interruzione risulta quasi impercettibile.
Nella prima fase di test abbiamo notato un ritardo superiore nel passaggio tra la clip 5 e la clip 6, il ritardo si aggirava attorno ai 276 ms contro i 40 di media. Abbiamo cercato di capire da cosa dipendesse. Ci siamo accorti

che nella visualizzazione dei risultati sul display al termine della riproduzione della clip 5 il sistema disegnava la barra di scorrimento per permettere di scorrere i risultati. Facendo in modo che l'operazione di aggiunta della barra di scorrimento fosse eseguita dopo l'inizio della riproduzione della clip Abbiamo eliminato questo ritardo.

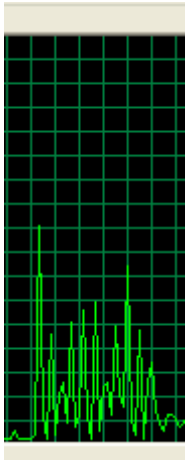
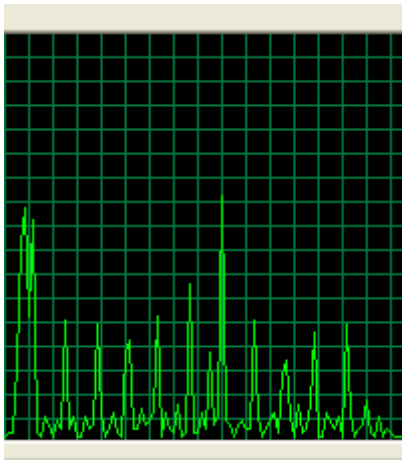
a)  b) 

Figura 6.7 - Overhead della CPU per a) 10 clip da 5 secondi
b) 10 clip da 15 secondi

Abbiamo volutamente riportato anche durate attorno ai 15 secondi, dove i tempi sono leggermente più alti, questo perché il dispositivo deve lavorare con file più grandi. La motivazione è che l'utilizzo di file di maggiore durata richiede un'attesa iniziale più lunga, ma permette interruzioni del collegamento più lunghe. Ad esempio un file di 15 secondi occupa nel formato scelto (linear, 11025Hz, 16 bit, mono) 323 KB di memoria. Il tempo di scaricamento si aggira mediamente intorno agli 8,4 secondi. Per le altre durate fare riferimento alla Tabella 6.1. A seconda delle condizioni può essere conveniente adattare la lunghezza delle clip.
Abbiamo misurato l'overhead del processore e come si vede dalla figura 6.7 le operazioni che richiedono maggiore utilizzo di cpu sono quello di inizializzazione e scrittura del file arrivando anche a punte di 35-40% dell'utilizzo del processore. Nelle fasi rimanenti di acquisizione si stabiliscono attorno a un 3-5%. Abbiamo riportato l'andamento per 10 clip da 5 secondi e 10 clip da 15 secondi.

6.6 Conclusione

In questo capitolo abbiamo parlato di ambiente di sviluppo e di fase di testing. Abbiamo visto quali strumenti ci hanno aiutato nello sviluppo di questa tesi e i dispositivi che ci hanno permesso di testare l'applicazione su dispositivi reali.

Abbiamo visto come funziona l'emulatore Wireless Toolkit della Sun e abbiamo provato la MIDlet su un pocket PC HP iPAQ h5500. Il collegamento bluetooth è stato possibile all'access point DBT-900AP, che ci ha permesso di utilizzare la servlet sviluppata su un server web Apache Tomcat.

Abbiamo quindi eseguito i test e riportato i risultati, notando che il risultato può essere considerato soddisfacente.

CONCLUSIONI

Il lavoro svolto in questa tesi ha avuto come obiettivo fondamentale la realizzazione di un progetto che consentisse la fruizione di contenuti multimediali su dispositivi mobili. Abbiamo visto come le limitazioni intrinseche di tali dispositivi abbiano portato ad utilizzare strumenti e ambienti diversi per lo sviluppo dell'applicazione. Abbiamo approfondito alcuni concetti relativi a J2ME, al Java Media Framework e alle Mobile Media API, necessari per sviluppare l'applicazione sia dal lato server che client.

L'applicazione realizzata propone un approccio innovativo per l'erogazione di un servizio di streaming su dispositivi mobili. Lo scopo era quello di rendere il sistema robusto a cadute di connessione approfittando dei momenti di connettività per avvantaggiarsi nel download delle parti della presentazione da eseguire. Abbiamo visto, anche con risultati sperimentali che eseguendo l'operazione di buffering riusciamo a garantire qualità e continuità della riproduzione, l'interruzione tra una clip e l'altra è quasi impercettibile e il vantaggio rispetto a uno streaming tradizionale è evidente. La mobilità è garantita e con il nostro approccio l'idea di sessione permette di decidere dopo disconnessioni prolungate da quale punto ripartire.

Nel corso del nostro lavoro ci siamo confrontati con problemi di risorse limitate e la robustezza della nostra architettura dipende anche da questo.

Si potrebbe pensare di estendere la nostra architettura anche per contenuti video, permettendo la visione di un film, di un concerto o di qualsiasi altra presentazione. Potrebbe essere anche pensato per essere usato come controllo di ambienti, ad esempio per un sistema di sorveglianza a distanza.

Bibliografia

[3GPP05] The 3GPP Home Page. 3GPP, 2005. http://www.3gpp.org

[AVWG05] Audio Video Working Group, http://www-mice.cs.ucl.ac.uk/multimedia/misc/avt/, 2005.

[ALM99] M. Albrecht, M. Frank, P. Martini, M. Schetelig, A. Vilavaara, A. Wenzel,"IP Services over Bluetooth: Leading the Way to a new Mobility", Proceedings of the 24th Conference on Local Computer Networks, Lowell, MA, October 1999, pages 2-11.

[BAS00] S. Baatz, M. Frank, et al., "Handoff Support for Mobility with IP over Bluetooth", IEEE Local Computer Network, pages 143-154, November 2000.

[BEP04] P. Bellavista, C. Stefanelli, M. Tortonesi, "Middleware-level QoS Differentiation in the Wireless Internet: the ubiQoS solution for Audio Streaming over Bluetooth", Proceedings of the First International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks (QSHINE'04), IEEE 2004.

[BES04] P. Bellavista, C. Stefanelli, M. Tortonesi, "The ubiQoS Middleware for Audio Streaming to Bluetooth Devices", Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04), IEEE 2004.

[BTM05] The Official Bluetooth Membership Site, https://www.bluetooth.org , 2005.

[BTS05] Bluetooth SIG, "Bluetooth specification", http://www.bluetooth.com.

[CHW03] W. Chan, J. Chen, P. Lin, K. Yen, "Quality-of-Service in IP Services over Bluetooth Ad-Hoc Networks", Mobile Networks and Application 8, pages 699-709, 2003.

[**CLDC**] http://java.sun.com/cldc/.

[COD05] D. Cotroneo, G. Paolillo, C. Pirro, S. Russo, "A User-driven Adaptation Strategy for Mobile Video Streaming Application", IEEE, International Conference on Distributed Computing System Workshops (ICDCSW'05).

[DEP05] Patricia McDermott-Wells, "What is Bluetooth?" IEEE Potentials, Vol. 23, Issue 5, Dec 2004-Jan 2005, pages 33-35.

[Eclipse] Eclipse Platform, http://www.eclipse.org/platform.

[EMEP] eclipseME, Eclipse Micro Edition Plugin, http://eclipseme.org.

[**J2ME**] http://java.sun.com/j2me/.

[JavOv] Java Technology Overview, http://java.sun.com/overview.html.

[JSR30] J2ME Connected, Limited Device Configuration http://jcp.org/en/jsr/detail?id=30.

[JSR37] Mobile Information Device Profile for the J2ME Platform http://jcp.org/en/jsr/detail?id=37.

[**JSR135**] Java Specification Request: Mobile Media API http://jcp.org/en/jsr/detail?id=135.

[KAA02] A. Kansal, U. B Desai, "Mobility Support for Bluetooth Public Access", IEEE international Symposium, Vol. 5, pages V-725-V-728, May 2002.

[**MID**] MID profile http://www.wmlscript.it/j2me/api20/index.html.

[**MIDP**] http://java.sun.com/products/midp/.

[**MMAPI**] Mobile Media API: http://java.sun.com/products/mmapi/.

[**MMAPI2**] http://java.sun.com/j2me/docs/alt-html/MMAPI-WP/mmapiwp4.html.

[**MMAPI3**] Mobile Media API Overview http://developers.sun.com/techtopics/mobility/apis/articles/mmapi_overview/.

[Mok98] http://www.mokabyte.it/1998/10/javastory.htm.

[MPEG05] MPEG Home Page, Moving Picture Experts Group, 2005. http://www.chiariglione.org/mpeg.

[PRA03] Amrit Prit Paul Singh Bilan, "Streaming Audio Over Bluetooth ACL Links" ITCC, pages 287-291, April 2003.

[Servlet02] Servlet API Documentation, The Apache Software Foundation 1999-2002, http://jakarta.apache.org/tomcat/tomcat-5.0-doc/servletapi/index.html.

[SYS04] Sysvideo Tomcat Launcher plugin 2004, http://www.sysdeo.com/eclipse/tomcatplugin.

[Tomcat05] The Apache Jakarta Tomcat 5 Servlet/JSP Container, The Apache Software Foundation 1999-2005, http://jakarta.apache.org/tomcat/tomcat-5.0-doc/.

[WT05] J2ME Sun Java Wireless Toolkit, http://java.sun.com/products/sjwtoolkit/index.html.

[WEME05] IBM WebSphere Everyplace Micro Environment 2005, http://www.ibm.com/software/wireless/wme/.

[ZES04] S. Zeadally, A. Kumar, "Protocol Support for Audio Streaming between Bluetooth Devices" IEEE, Radio and Wireless Conference, pages 303-306, September 2004.