UNIVERSITA' DEGLI STUDI DI BOLOGNA

FACOLTA' DI INGEGNERIA

Corso di Laurea in Ingegneria Informatica

Reti di Calcolatori

SERVIZI MULTIMEDIALI

PER SISTEMI WIRELESS

Tesi di laurea di:

Vittoria Caranna

Relatore:

Chiar.mo Prof. Ing. **Antonio Corradi**

Correlatore:

Dott. Ing. **Luca Foschini**

Anno Accademico 2003-2004

INTRODUZIONE

Grazie allo sviluppo e alla diffusione delle tecnologie wireless gli ultimi anni sono stati caratterizzati da una sempre maggior richiesta di servizi accessibili ovunque. L'erogazione di queste nuove tipologie di servizio ha portato a concepire in modo nuovo i sistemi distribuiti in modo da assistere i continui movimenti delle persone.

La maggior parte delle applicazioni è sviluppata sul modello di comunicazione cliente/servitore ma, per venire incontro alle esigenze richieste e cioè alla possibilità di reperire le informazioni su un qualunque dispositivo, si è dovuto ampliare questo modello con delle soluzioni che permettano di realizzare dei modelli più avanzati. Occorre realizzare degli schemi di comunicazione che permettano agli utenti di reperire le informazioni in relazione al luogo in cui essi si trovano.
Negli ultimi anni si è assistito allo sviluppo di applicazioni multimediali sempre più all'avanguardia, come è accaduto alla telefonia mobile. I telefoni cellulari sono quelli che commercialmente hanno ottenuto un successo tale da eguagliare i livelli di utenza della telefonia fissa. Accanto a questi si sono affiancati computer palmari che possono garantire prestazioni comparabili a quelli di un piccolo personal computer. Per venire incontro alle esigenze di un'utenza che chiede il reperimento delle informazioni su un dispositivo wireless, si è cercato di appoggiarsi il più possibile a questi dispositivi, tenendo conto delle problematiche connesse a questi device e, in particolare, ai limiti che essi presentano.
Grazie a questi dispositivi è stato possibile realizzare un tipo di comunicazione più sofisticata ma, si sono dovuti affrontare degli inconvenienti legati ad essi in quanto tali applicazioni presentano dei grossi limiti per la richiesta di presentazioni multimediali e la gestione di esse. Non esistono, infatti, degli strumenti come i dispositivi wireless che siano in grado di gestire grandi quantità di dati, come richiesto dall'erogazione di servizi multimediali.
Il lavoro sarà basato sulla realizzazione di un software che permetta di richiedere e gestire dei flussi multimediali e la possibilità di effettuare uno streaming da un web server su client leggeri come i telefoni

cellualri, cercando di realizzare la trasmissione dei dati con delle modalità che differiscono dalle usuali e che consistono nella possibilità di visualizzare dei dati multimediali senza che essi siano stati salvati interamente su locale. Il lavoro di questa tesi affronta tale problematica con l'intento, in particolare, di gestire dei flussi multimediali in modo da offrire un supporto all'utilizzo delle risorse e facilitare lo sviluppo di nuove applicazioni che siano efficaci alla mobilità degli utenti. Per gestire la trasmissione e il reperimento delle informazioni da un web server, verranno implementate delle funzionalità sul lato server che permettono di gestire la presentazione multimendiale e di manipolarla secondo le esigenze di realizzazione del progetto. Il client dovrà gestire la ricezione e la rappresentazione del media in modo da far fronte a temporanee cadute di connessione a da non dover scaricare tutta la presentazione che viene richiesta dall'utente e, per realizzare tali funzionalità verranno effettuate delle implementazioni adatte .
Verranno implementati dei servizi che saranno resi accessibili alle applicazioni secondo le esigenze del cliente.

La tesi è così organizzata:
nel primo capitolo, si passa alla descrizione delle applicazioni multimediali e al legame che esse hanno con i dispositivi wireless, mettendo in evidenza i loro limiti, in particolare quelli relativi alla loro memoria e alla possibilità di gestirla in maniera adeguata per gli scopi realizzativi del progetto. Si evidenziano anche le scelte che hanno spinto ad effettuare una tale organizzazione del progetto, con le motivazioni che hanno inciso su tale direzione che è quella di poter realizzare una trasmissione basata su uno streaming. Sempre in questa prime parte, si descrive la differenza che c'è tra il tipo di trasmissione che è stata affrontata e realizzata in questo progetto e quella tipica che solitamente viene utilizzata per effettuare il trasferimento di un media che vede la memorizzazione di tutta la risorsa in locale prima di una possibile visualizzazione della stessa.
Nel secondo capitolo si passa alla descrizione dei package della piattaforma J2ME dedicati alla programmazione di rete, ed in particolar modo, la nuova estensione MMAPI dedicata alla programmazione di applicazioni multimediali. Si passa alla descrizione di un MIDlet e delle fasi che caratterizzano il suo percorso di creazione e realizzazione. Parlando delle api della JMF, introdotte per esplicare la realizzazione

del lato server ed in particolare, per estender le sue funzionalità, si tiene conto anche delle differenze e delle somiglianze con il package delle MMAPI, utilizzate invece per supportare i limiti dei dispositivi wireless e per realizzare le funzionalità richieste a tali dispositivi.
Nel terzo capitolo si parla della scelta architetturale del progetto, per poi scendere nella descrizione dell'architettura di ciascuna entità e cioè del client e del server. Si darà anche una descrizione delle fasi di comunicazione previste tra le due entità e cioè, la fase di accordo iniziale che vede coinvolta la socketConnection e la fase di trasferimento della risorsa, realizzata con l'ausilio del protocollo http.
Il quarto capitolo sarà dedicato alla descrizione di un'applicazione per lo streaming video, alle scelte implementative che sono state adottate sia per il lato server che per il lato client, sottolineando il ruolo svolto dai threads nella realizzazione del progetto.
Nel quinto capitolo si passa alla descrizione del Wireless Toolkit che è l'emulatore utilizzato per la fase di testing e si analizza il risultato del test effettuato sul progetto, evidenziando le caratteristiche che si possono evincere da quest'ultimo.

Capitolo 1

Streaming Multimediale su dispositivi wireless

1.1 Tematica del progetto

In questo capitolo si vuole concentrare l'attenzione su quale problematica si sta affrontando, mettendo in evidenza le questioni che le fanno da contorno e le scelte che sono state adottate per una possibile soluzione al problema.
La tematica che viene affrontata in questa tesi riguarda l'esigenza di un'utenza che richiede sempre più la possibilità di reperire le proprie notizie, informazioni da qualunque luogo si trovi, su un dispositivo diverso dal quale stava, ad esempio, operando in precedenza. Per venire incontro a questa richiesta, si è pensato alla realizzazione di un software che faccia da spunto per una possibile soluzione al problema e che si appoggia ad una questione che coinvolge i dispositivi wireless. Si è voluto realizzare un'applicazione che permetta una trasmissione di media su un dispositivo wireless secondo una modalità diversa da quella tipica. Questa modalità consiste nella possibilità di poter sfruttare prima di tutto il protocollo http che è già supportato dai dispositivi wireless, cosa che non accade ancora per il protocollo rtp; sfruttando il protocollo http, si può reperire un media presente su un web server e scaricarlo in locale senza la necessità di salvarlo per effettuarne la visualizzazione su display e il risultato che si ottiene consiste nel realizzare il merging dei vari pezzi costituenti il media, il cui spezzettamento è stato effettuato grazie ad una estensione delle funzionalità sul lato server.
Verrà presa in esame la problematica relativa alla garanzia di una continua visualizzazione a fronte di disconnessioni wireless; perché ciò sia garantito, verrà data una spiegazione della scelta organizzativa del progetto, sottolineando il fatto che nonostante lo scaricamento del materiale multimediale avvenga utilizzando tecnologie standard, come il protocollo http, non vi sia la necessità, lato utente, di avere scaricato tutta la presentazione prima di presentarla. Verrà dunque presa in esame la differenza tra la modalità di trasmissione standard e la modalità realizzata.

Poi verranno presentati i concetti di applicazione multimediale, di streaming e la relativa realizzazione verso dispositivi di memoria limitata. A questo segue una spiegazione delle scelte organizzative del progetto, cioè si mette in evidenza come il progetto è sviluppato e di seguito implementato ma si danno anche delle chiare motivazioni che hanno condotto ad una tale scelta.

1.1 Trasmissione multimediale "standard"

Cominciamo adesso con il mettere in evidenza quelle che sono le differenze relative ad una modalità di trasmissione standard e quella che invece è stata realizzata nel progetto, sottolineando il coinvolgimento che implica un'applicazione multimediale.
Le applicazioni multimediali devono far fronte alle ingenti richieste di risorse e alla relativa garanzia di qualità di servizio che richiede una trasmissione della quale si vuole un esito positivo.
Una trasmissione cliente/servitore ***standard*** prevede delle fasi e delle metodologie di trasmissione che non sono favorevoli alle richieste di un'utenza che esige il reperimento delle informazioni in base alla località.
Secondo tale tipologia di trasmissione di una risorsa si fa uso di un protocollo TCP/IP e si presume che tutta la risorsa sia stata trasferita, in maniera completa, prima di poter essere utilizzata dall'utente.
Quindi, nello sviluppo di un'applicazione multimediale occorre tenere in considerazione dei limiti che possono essere imposti dal protocollo di rete utilizzato, dalla carenza di risorse fisiche (CPU, memoria).
Finché Internet era usato per trasmettere informazioni come file, posta, non vi erano problemi di QoS. Tali problemi diventano evidenti con l'avvento di protocolli che permettono di usare Internet per trasmettere informazioni audio e video. Secondo tale modello, si rimane un po' vincolati ai limiti fisici di un dispositivo con la seguente conseguenza di impossibilità nell'effettuare determinate richieste e servizi che sono ormai indispensabili e richiesti da un numero sempre più elevato di utenza. Non solo si rimane vincolati ai limiti dei dispositivi fisici ma non si riesce ad ottenere dei risultati soddisfacenti in termini di efficienza di trasmissione, in quanto si possono avere possibili ritardi o perdite della risorsa stessa di fronte a malfunzionamenti che si possono verificare nel corso della trasmissione.

Passiamo adesso ad evidenziare alcuni di questi limiti che scaturiscono nell'uso di questa tipologia di trasmissione.

1.1.1 Limiti del protocollo IP

Il protocollo IP(*Internet Protocol)* è un protocollo "best-effort" che permette ai pacchetti che devono viaggiare da una sorgente ad una destinazione di passare attraverso dei nodi intermedi senza lasciare informazioni sul tragitto compiuto in precedenza.
Una volta che il pacchetto è stato spedito, il mittente non ha modo di conoscere se il destinatario lo ha ricevuto.
Si evince che il protocollo IP non garantisce affidabilità di trasmissione. Per ovviare a questa mancanza, sopra l'IP esiste un altro protocollo, il TCP, che ha come compito anche quello di ritrasmettere un pacchetto quando ritiene che questo non sia giunto a destinazione.
Questo protocollo assicura però che i pacchetti non giunti a destinazione vengano ritrasmessi, ma non assicura efficienza, né offre garanzia sui tempi di consegna.
Da una parte, comunque, la rete che si basa su questo protocollo risulta più semplice, dall'altra bisogna considerare che esso non è nato per gestire servizi che richiedono una certa qualità di servizio.

1.1.2 Limiti delle risorse fisiche

I limiti a cui occorre far fronte nel momento della gestione di un'applicazione sono inerenti alla memoria, alla banda trasmissiva e alla CPU.
Per quanto riguarda la ***memoria***, essa è richiesta in maniera elevata per la trasmissione e la visualizzazione di una presentazione multimediale, in particolar modo se la presentazione deve essere memorizzata prima di poter essere visualizzata. Questa operazione di bufferizzazione dei dati è utile quando occorre mantenere la visualizzazione nei casi ritardi trasmissivi, o per memorizzare i dati nelle fasi di codifica e decodifica.

La ***CPU*** è la risorsa che viene richiesta in maniera efficiente nel momento in cui essa deve gestire l'applicazione multimediale. Si occupa, infatti, nel momento della ricezione dei dati dello spacchettamento, della decodifica e del successivo *rendering*. Nel caso

in cui i dati devono essere inviati, si occupa della codifica e dell'impacchettamento.
Da questo si evince come questa risorsa sia sottoposta ad un carico di lavoro non indifferente e cha la prerogativa delle azioni portate a termine dipenda in maniera consistente da essa.

E' stato evidenziato che la ***banda*** trasmissiva richiesta per il trasferimento di un'applicazione multimediale debba essere abbastanza elevata, ma vi sono delle tecniche per ovviare a tale inconveniente; si può far uso della compressione dei dati che riduce la larghezza di banda utilizzabile, e nel caso dei video è possibile abbassare tale valore grazie alla codifica MPEG.

1.2 Streaming multimediale

Si passa ad evidenziare gli aspetti legati alla tipologia di trasmissione che permette di effettuare il trasferimento di una risorsa come file, con il supporto di un protocollo come l'http. Tale tipo di trasmissione è stata realizzata in questo progetto con l'intento di venire incontro alle esigenze che si vogliono soddisfare, senza trascurare quelli che sono i limiti dei dispositivi presi in questione ma, anzi, cercando di affrontarli e superarli, utilizzando al meglio le funzionalità che si possono sfruttare da questi device mobili.

1.2.1 Caratteristiche rilevanti

Per **streaming** s'intende un **flusso di dati** il cui profilo generato deve essere uguale al profilo da produrre. Lo streaming permette l'interattività con altre persone o con un altro sistema e tutto ciò avviene in tempi di risposta molto brevi.
Oggetto preso in esame sarà, prima di tutto, una ***presentazione multimediale*** con la quale s'intende l'utilizzo contemporaneo di vari **media** come *video, immagini, suoni, testo;* ciò costituisce un insieme di uno o più oggetti multimediali dotati di un concetto di qualità.
Per ***oggetto multimediale*** s'intende l'astrazione di un'istanza di un certo tipo di dato multimediale. Ogni oggetto multimediale può incapsulare un solo tipo di dato, cioè può essere solo di tipo audio, video, ecc…A ciascun oggetto multimediale corrisponde un file che viene salvato sul

lato server e può essere reperito ad un destinatario seguendo un percorso che chiameremo path.

Ora si è interessati ad un procedimento in cui non viene salvata tutta la risorsa in locale e porta alla fruizione da remoto del flusso dati.
Questo tipo di scelta è funzionale e adatta quando l'accesso avviene da una rete mobile, cioè attraverso un device di tipo wireless e si vuole che l'infrastruttura sia organizzata per seguire i movimenti dell'utente.
Non è da sottovalutare la relativa gestione delle risorse supportando la prenotazione e il monitoraggio delle stesse. Grazie a questa scelta non solo si viene incontro alle richieste di un'utenza che esige sempre più dei miglioramenti nel campo informativo ma, come già è stato evidenziato, si riesce a supportare una certa garanzia di qualità nella trasmissione in caso di possibili malfunzionamenti nella rete o di cadute di connessione wireless. Nonostante si possa cadere in questi inconvenienti, si ha la possibilità di gestire la situazione che è insita nella progettazione del software stesso, grazie al quale si evitano intasamenti nella rete dato che la risorsa non viene inviata tutta completa intesa come unico blocco, ma in varie tracce che la costituiscono, evitando ritardi o malfunzionamenti della stessa.

Figura 1: streaming da server

Per poter realizzare gli aspetti che sono stati appena evidenziati, non ci si basa solo sulla scelta realizzativa del progetto, ma sono compresi anche i vari strumenti che fanno parte della realizzazione e che sicuramente contribuiscono alla buona riuscita.
E'opportuno sottolineare il ruolo svolto dalle entità in gioco nel progetto che qui vengono riportate.

1.3 Applicazioni multimediali e dispositivi wireless

Per affrontare al meglio la problematica che è stata appena accennata, verrà data una semplice trattazione di quello che sono le applicazioni multimediali e come esse sono legate ai dispositivi wireless, cioè come è possibile effettuare una visualizzazione delle stesse sfruttando quelli che sono i limiti dei device mobili.
Un'applicazione multimediale viene usata principalmente per manipolare informazioni di varia natura e fornirle all'utente. Le informazioni vanno da tipologie più semplici, come testo, a tipologie più complesse come video, immagini, animazioni e grafici. Alcuni esempi di applicazioni multimediali possono essere le applicazioni per il video on demand, i sistemi per le video-conferenze.
L'informazione che viene fornita consiste in una sequenza di campioni che hanno una certa dipendenza temporale tra di loro; con il termine *stream* s'intende una sequenza di campioni che costituiscono un'informazione continua.

1.3.1 Aspetti dei dispositivi wireless

Ai nostri giorni, i dispositivi wireless hanno preso il sopravvento e dunque lo studio che viene affrontato su di essi riguarda la possibilità di migliorare la gestione delle limitate risorse disponibili. Occorre dare uno sguardo al fatto che essi hanno sì una grande portabilità e dei vantaggi legati alla possibilità di un loro utilizzo in un qualunque posto, tutto ciò possibile grazie alle loro dimensioni ma è proprio per questo che essi non possono disporre di un'elevata capacità di memorizzazione e di computazione. Questo aspetto verrà preso in esame e verrà sottolineato

il fatto che gli studi riguardano la possibilità di sfruttare la memoria che essi hanno a disposizione per poter realizzare delle funzionalità che senza delle tecniche apposite non potrebbero essere realizzate. Uno sguardo viene dato proprio alla problematica in questione e cioè alla possibilità di poter effettuare la visualizzazione di una presentazione, ad esempio un video, che viene scaricata da un web server, sfruttando solo la memoria di cui tali device dispongono, senza dare limiti sulle dimensioni della risorsa che si vuole ottenere.
Per poter gestire tale funzionalità, è stato adottato un approccio apposito che consiste in una diversa modalità di trasmissione del media stesso, potendo così permettere la visualizzazione dello stesso su un dispositivo, pur non dovendolo mantener per intero sul device mobile.
Adesso verrà descritto come è stato possibile gestire in maniera efficiente la memoria che tali dispositivi dispongono, riuscendo ad ottenere dei risultati promettenti sugli obiettivi posti.

1.4 Gestione della memoria

Si passa adesso a dare una descrizione di come è stata utilizzata la memoria dei device wireless per gli scopi che ci siamo posti e cioè quelli riguardanti la possibilità di scaricare su tali dispositivi delle risorse da un web server, mediante un approccio apposito, e di poterne effettuare la visualizzazione.
Nel progetto in esame, la problematica consiste nel riuscire a gestire una risorsa, che arriva da un web server, ed effettuarne la visualizzazione sul display di un device dalle capacità limitate, come i telefoni cellulari. Non si può pensare di memorizzare tutta la risorsa su locale perché ciò implica che il dispositivo abbia memoria a sufficienza e in più non si garantisce una buona qualità di trasmissione nel trasferimento della risorsa stessa. Infatti, volendo garantire che l'utente abbia una corretta visualizzazione del media su display e dovendo venire incontro ai limiti dei dispositivi sui quali si opera, si è adottata una scelta implementativa che ha permesso di scaricare la risorsa da un web server senza la necessità di salvarla in locale, ma di ottenerla in piccole parti volta per volta e di realizzare una visualizzazione della stessa in maniera continua grazie al merging delle varie tracce. Adottando questa modalità di trasmissione si viene incontro in maniera esaustiva ai limiti imposti dai dispositivi wireless, in quanto dato che la risorsa non deve essere

memorizzata in locale prima di poterne effettuare la visualizzazione, si può scegliere di trasmettere anche dei media di una certa dimensione.
La scelta di questo approccio implementativo permette la realizzazione di un protocollo di trasferimento grazie al quale si può sfruttare la memoria di un dispositivo wireless senza avere dei limiti circa le dimensioni del media. Questo è possibile grazie alle funzionalità sviluppate sul server che permettono, appunto , di render disponibile la risorsa come file. Vedremo adesso le motivazioni che hanno portato alla scelta di questa implementazione.

1.5 Scelte organizzative

Una volta che sono stati evidenziati i limiti che caratterizzano i dispositivi wireless legati alla disponibilità e gestione della memoria, si vogliono sottolineare anche altre motivazioni che hanno indotto ad effettuare il tipo di scelta organizzativa del progetto.
E' stato detto che la modalità di trasferimento di un media da un web server è stata realizzata sfruttando la possibilità di disporre sul web server stesso di una presentazione sotto forma di file di cui sono presenti vari pezzi e la cui realizzazione e gestione è demandata al server stesso. Disponendo di una risorsa così sul web server, la cui richiesta viene comunque avanzata da un possibile utente, la trasmissione della stessa può avvenire pezzo per volta, con il supporto di un protocollo di comunicazione che ne renda possibile il trasferimento. Riuscendo a gestire una tale trasmissione, che non implica la memorizzazione del file in locale prima della visualizzazione, non solo non si condizionano le risorse del device alla dimensione del media, ma si riesce a gestire anche una certa qualità nella trasmissione. Prima di tutto si ha la possibilità di fare il merging delle varie parti che consiste nel far partire la visualizzazione della prima non appena arriva in locale e, mentre ciò avviene, si continua a reperire le altre e che verranno eseguite in modo da rendere il filmato in maniera sequenziale e continua, cercando di dare all'utente una visualizzazione nella quale non si percepiscano le cuciture fra i pezzi presentati. Per dare dei risultati ottimali, si è cercato di realizzare l'applicazione in modo tale da effettuare un merging delle parti in maniera il più ottimale possibile. La realizzazione del progetto è basata sull'utilizzo di strumenti standard come un server web, opportunamente estesi, con la funzionalità adatta per spezzare i video, e

protocollo http per il trasferimento file. E' da sottolineare che il protocollo http è supportato dai dispositivi cellulari, mentre altri protocolli più specifici, come l'rtp, possono non esserlo.

1.5.1 Vantaggi delle scelte organizzative

Sono stati evidenziati gli aspetti che caratterizzano la realizzazione del progetto e si è messo in evidenza come esso è stato realizzato, sottolineando che grazie a tale realizzazione si è riusciti a soddisfare la richiesta di trasferimento di media su dispositivi dalle capacità limitate ma adesso si vogliono evidenziare altre motivazioni che hanno portato ad effettuare una tale scelta implementativa. Realizzando il progetto mediante un protocollo che permette di effettuare il trasferimento di una risorsa in varie parti, ed effettuando il merging sul lato client, permette di trarre altri vantaggi legati ad aspetti qualitativi nella trasmissione di una risorsa. Grazie a questa realizzazione che permette di poter effettuare un trasferimento della risorsa traccia per traccia, evita possibili situazione di congestione della rete in situazioni di caduta della connessione wireless.
Infatti, anche di fronte a possibili brevi cadute di connessione, l'utente ha la garanzia di poter continuare a vedere il media su display perché il materiale presentato è salvato localmente. Potremmo anche mettere in evidenza che in caso di caduta durante il trasferimento si può cominciare da capo, o meglio riprendere il trasferimento da dove si era rimasti. Questo è realizzabile grazie a dovuti accorgimenti e realizzazioni che permettono di effettuare un controllo sul trasferimento, tenendo traccia del punto in cui si è giunti, per poter effettuare possibili ritrasmissioni in caso di avvenute disconnessioni. Lo streaming multimediale che avviene è possibile grazie al supporto che sta alla base della realizzazione e dagli aspetti appena descritti e di cui si può trarre vantaggio, si può dire che le motivazioni che hanno portato ad una tale scelta implementativa sono abbastanza valide e soddisfacenti. E' utile sottolineare come una scelta sia legata non solo alla buona riuscita di un progetto, ma alle motivazioni che hanno spinto ad affrontarla perché da queste si possono trarre dei benefici legati a diversi aspetti, alcuni dei quali sono quelli appena descritti.

Si passerà adesso a chiarire meglio il funzionamento della modalità di trasmissione, evidenziando la differenza tra la modalità standard e quella che fa uso dello streaming multimediale.

Capitolo 2

J2ME e MMAPI

2.1 La famiglia Java

Il linguaggio di programmazione Java ha avuto negli anni una grande diffusione prima nel mondo client, attraverso la realizzazione di applet, poi nel mondo server, grazie alle tecnologie servlet ed EJB (Enterprise Java Beans), che hanno permesso la realizzazione di servizi WWW e sistemi di back-end.
La Sun ha creato tre diverse "edizioni" (edition) di Java, che pur rimanendo basate su virtual machine compatibili ed un insieme di API di base, si sono differenziate a seconda del loro target.
Le "edition" sono state introdotte con il rilascio di Java2 e dividono piattaforme, API, e strumenti di sviluppo in tre differenti gruppi ognuno dei quali indirizzato ad un ben preciso segmento di mercato.

° *Standard Edition J2SE*: è la versione base di Java; essa viene utilizzato sviluppare software destinato a workstation e stazioni desktop.

° *Java 2 Enterprise Edition J2EE*: fornisce un insieme di API per la realizzazione di architetture solide, complete e scalabili, dirette, soprattutto, ad applicazioni server.

° *Java 2 Micro Edition J2ME*: è l'ultima piattaforma ideata da Sun studiata appositamente per i dispositivi embedded ed, in particolar modo, per apparecchiature che hanno risorse limitate.

2.2 Piattaforma J2ME

Nel giro di diversi anni, le capacità del linguaggio di programmazione Java e le sue potenzialità come piattaforma per applicazioni free-standing, hanno mosso l'interesse dei programmatori come via per ridurre i costi di uno sviluppo software.

La Sun ha espanso le dimensioni della piattaforma Java e questa espansione include un complesso set di librerie di interfacce utente che sono utili a questi ultimi per poter creare applets e anche per migliorare l'efficienza e la sicurezza.
Con il passar del tempo, la Sun ha rilasciato la piattaforma Java2, e da allora è divenuto necessario separarla in diverse parti. Il cuore della funzionalità sta nel fatto che il minimo supporto richiesto per ogni ambiente Java sia impacchettato come java 2 Standard Editon (JESE).
Molti pacchetti opzionali possono essere aggiunti alla JESE per soddisfare specifiche richieste per una particolare applicazione, come una sicura estensione socket. La Sun risponde anche a un crescente interesse nell'uso di Java per sviluppi enterprise-level e per ambienti di applicazioni server con la Java 2 Enterprise Edition (J2EE), che incorpora la nuova tecnologia delle servlets, , Enterprise JavaBeans, e JavaServer pages.
Mentre gli sviluppi di Java sono stati proiettati verso Internet, la domanda ultimamente cresce per l'utilizzo di java su piccoli dispositivi. La Sun allora risponde creando piattaforme Java dalle capacità ridotte. Queste piattaforma sono basate sul JDK 1.1, il predecessore della piattaforma Java 2, e propongono differenti approcci al problema di ridurre la piattaforma ad essere capace di adattare le risorse disponibili. Ciascuna di queste piattaforme rappresenta una soluzione ad-hoc al problema.
La J2ME è una piattaforma per piccoli dispositivi, con la quale s' intende eventualmente poter sostituire i vari prodotti JDK 1.1 con delle su lozioni basate su Java2. Il micro-world è caratterizzato da un range abbastanza vasto di dispositivi aventi differenti capacità e per questo non è possibile creare un unico prodotto software che sia adatto per tutti. La J2ME è una collezione di specifiche che definiscono una serie di piattaforme, ognuna delle quali è adatta per un subset dell' insieme totale dei dispositivi. Il subset dell'ambiente totale di programmazione Java per un particolare dispositivo è definito da uno o più *profili*, che estendono le capacità base delle *configurazioni*. La configurazione e i profili che sono appropriati per un dispositivo dipendono entrambi dalla natura dell'hardware.

2.3 Architettura di un'applicazione J2ME

L'architettura di un'applicazione sviluppata sulla piattaforma Micro Edition risulta essere leggermente più complessa rispetto ad una normale applicazione, dal momento che entrano in gioco i concetti di *Configurazione* e di *Profilo.*
Per sistemi che aderiscono allo standard CDC, il sistema operativo si trova a contatto con la *normale* Java Virtual Machine mentre nel caso dei CDLC si tratterà della KVM o di qualsiasi altra conforme alle specifiche riportate in CLDC. Sopra alla macchina virtuale agiscono le librerie delle Configurazioni che collaborano con le API dei Profili. Infine vi sono i vari package opzionali che forniscono al programmatore specifiche classi per risolvere determinati problemi. Attualmente il maggiore impiego della piattaforma J2ME è dovuto alla programmazione di sistemi appartenenti a CLDC che sfruttano il profilo MIDP cioè telefoni cellulari, computer palmari e cercapersone.

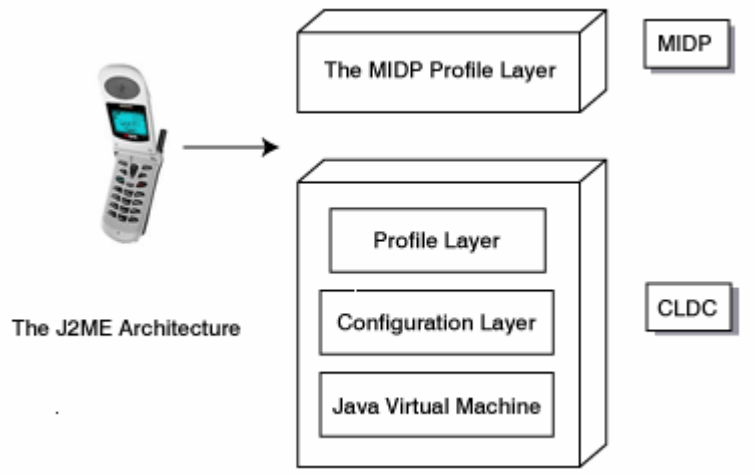

Figura 2: architettura J2ME

2.3.1 Virtual Machine

Le applicazioni Java devono la loro funzionalità ad una macchina che è la *Java Virtual Machine (JVM)*. Essa ha la funzione di svolgere molti compiti tra i quali la sicurezza, la gestione di thread, l' allocazione della memoria e oltre a tutto ciò traduce i file sorgenti in linguaggio binario comprensibile alla macchina sulla quale è in esecuzione l'applicazione.

Per i dispositivi che fanno parte della famiglia CDC non ci sono differenze di VM, infatti la macchina virtuale ha le stesse specifiche di quella utilizzata con J2SE. Invece, per gli apparecchi CLDC, nei quali i limiti relativi alla memoria sono più marcati, Sun ha sviluppato un'implementazione di riferimento, conosciuta con il nome di *K Virtual Machine (KVM).* Tale macchina virtuale è stata ideata per gestire in maniera opportuna gli aspetti speciali dei dispositivi a risorse limitate.
E' da sottolineare il fatto che l'implementazione di Sun di KVM non è l'unica possibile macchina virtuale di questo tipo, infatti CLDC riporta i requisiti fondamentali di cui deve disporre una virtual machine per offrire le funzionalità necessarie ai dispositivi che rientrano nella categoria CLDC.

2.3.2 Concetto di Configurazione

La Configurazione è una specifica che definisce l'ambiente software per un range di dispositivi definiti da un set di caratteristiche.
La Configurazione è strettamente legata alla Java Virtual Machine ed essa è costituita da una serie di funzionalità del linguaggio Java e dalle librerie principali della JVM, che possono essere necessarie per implementare le funzionalità richieste da un certo gruppo di dispositivi. I compiti software attribuiti alla Configurazione sono legati alla corretta gestione dell' hardware, infatti ogni particolare Configurazione si occupa della gestione memoria, gestione dello schermo, controllo delle risorse di rete e controllo della CPU.

La J2ME definisce due configurazioni:

Connected Limited Device Configuration (CLDC)

La CLDC è tenuta ad un basso livello dai consumatori elettronici. Una tipica piattaforma CLDC è una cella telefonica o una PDA con 512 KB di memoria disponibile. Per questo motivo, il CLDC è associato al wireless java, che riguarda la possibilità che ha un utente di poter scaricare piccole applicazioni java, note come *MIDlets*.

Connected Device Configuration (CDC)

La CDC coglie i bisogni dei dispositivi che stanno tra quelli che hanno una configurazione CLDC e i sistemi che girano sulla J2SE. Questi dispositivi hanno più memoria (tipicamente 2 MB o più) e dei processori molto più potenti, e possono come tali supportare un completo ambiente java.

Ogni configurazione consiste di una java virtual machine e di una collezione di classi java che fornisce l'ambiente di programmazione per le applicazioni software. Le limitazioni di memoria, specialmente nei dispositivi come i cellulari, possono rendere impossibile ad una virtual machine J2ME di supportate tutte le specifiche di un linguaggio Java o le istruzioni byte codes e le ottimizzazioni software previste da una VM J2SE.
Dove possibile, la J2ME utilizza le classi e i package J2SE.

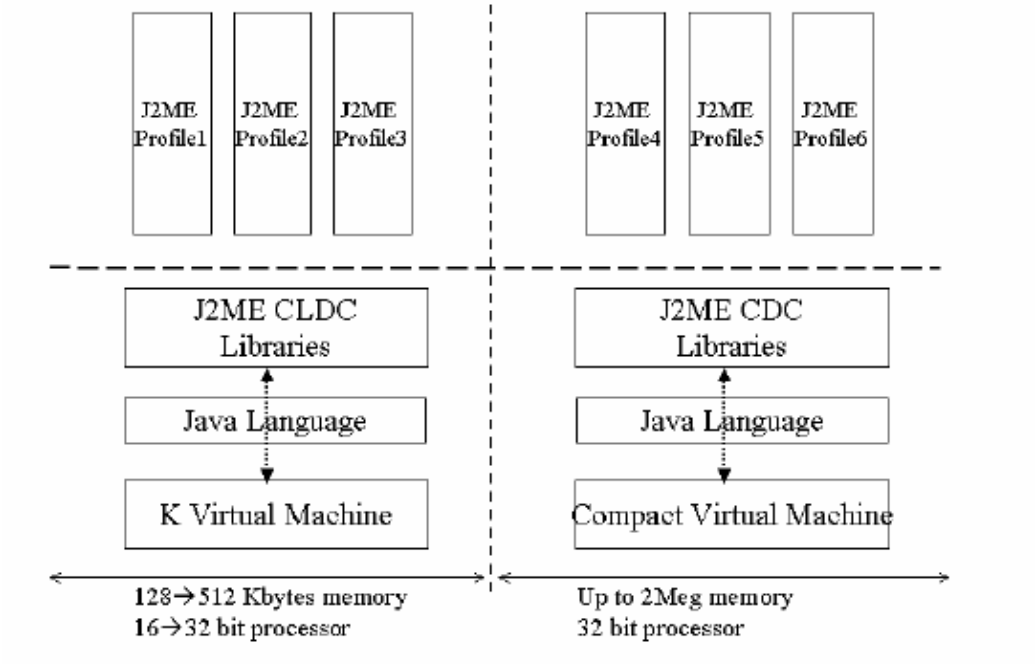

Figura 3: realizzazione del CLDC e del CDC

2.3.3 Concetto di Profilo

Il Profilo estende la Configurazione, infatti esso fornisce una serie di librerie che permettono ai programmatori di scrivere applicazioni per dispositivi particolari o per particolari segmenti di mercato.

Mobile Information Device Profile (MIDP)
Questo profilo aggiunge componenti di rete, interfaccia utente e storage locale al CLDC. Questo profilo prende in considerazione le limitazioni dello schermo e della memoria degli apparecchi mobili, fornendo un'interfaccia utente, le funzioni di rete basate su http 1.1 e i timer.

PDA Profile (PDAP)
Il profilo PDA è simile al MIDP, ma è rivolto a quei dispositivi che hanno più memoria. Il profilo PDA offre una più sofisticata libreria e delle api Java-based per accedere ai punti del sistema operativo.

Foundation Profile
Il profilo foundation estende la CDC per includere i package principali della J2SE 1.3; è molto significativo in quanto costituisce la base di appoggio per gli altri profili.

Personal Basis and Personal Profile
Questo profile aggiunge le funzionalità di base per la programmazione dei componenti di interfaccia utente.
Lo scopo di questo profilo è di essere utilizzato su quei dispositivi CDC che dispongono di schermi semplici, senza eccessive funzionalità e che dunque non permettono la visualizzazione di più finestre contemporaneamente.

RMI Profile
Il profilo RMI aggiunge le librerie per il Remote Method Invocation al profilo Foundation permettendo la comunicazione con la piattaforma J2ME.

Game Profile

Ha come target delle console per videogiochi facenti riferimento a dispositivi con una elevata memoria e con 32-64 MB di RAM e grafica 3D accelerata. Anche se questi dispositivi sono assimilabili a dei computer tradizionali, si è pensato che il profilo J2SE sia comunque troppo pesante (tenendo conto del fatto che presenta delle API che sono inutili in questo ambiente), anche se in questi tipi di dispositivi si tende a sfruttare al massimo l'hardware.

L'utilizzo dei package è in realtà molto flessibile: un dispositivo può supportare diversi profili, è dunque il programmatore che sceglierà quello che ritiene più idoneo e conveniente per una data applicazione.
Talvolta infatti un profilo, seppure dedicato ad una categoria di dispositivi differente, può risultare particolarmente adatto alla realizzazione di un applicativo (utilizzo "orientato all'applicazione"), quando è invece necessario utilizzare le caratteristiche proprie dei vari dispositivi è necessario ricorrere al profilo dedicato (utilizzo "orientato al dispositivo").
Occorre aggiungere che, per le loro caratteristiche intrinseche alcuni profili risultano più "device-specific" (es. PDA Profile, MID Profile) di altri (es.Foundation Profile, PersonalProfile, RMI Profile) che possiamo definire invece "application-specific".

Per adesso, il profilo che ha maggiore impiego è il profilo MIDP poiché è la base del Wireless Java.

2.4 Connected Limited Device Configuration

Il *Connected Limited Device Configuration* (CLDC) è il mattone base sul quale i profili J2ME per dispositivi di piccole dimensioni, come telefoni cellulari, sono costruiti. Questi dispositivi sono caratterizzati dalla loro limitate risorse di memoria e ciò rende impossibile per essi la possibilità di ospitare una piattaforma Java. Il CLDC specifica un set di package Java, delle classi e una ridotta funzionalità della java virtual machine che può essere implementata nelle risorse messe a disposizione dai dispositivi.

Le limitazioni hardware e software imposte dai dispositivi per i quali il CLDC è impiegato rende impossibile supportare in maniera soddisfacente una JVM o un set completo di classi.
I requisiti minimi di cui occorre disporre per il CLDC sono:
° 128 KB di ROM, per un salvataggio persistente della VM e delle librerie che caratterizzano la piattaforma CLDC;
° 32 KB di memoria volatile utile per l'allocazione runtime. Questa memoria è usata per soddisfare i requisiti dinamici delle applicazioni Java, il che include il caricamento delle classi e l'allocazione dello spazio nell'heap per gli oggetti e lo stack.
Per supportare un ambiente java runtime con queste risorse limitate, il CLDC definisce dei requisiti limitati per la virtual machine. Piuttosto che sui requisiti di memoria, il CLDC fa delle assunzioni sulla piattaforma. Per esempio, non assume che il dispositivo avrà un particolare tipo di display o un meccanismo di input per l'utente come il mouse, e non richiede alcun tipo di storage locale per i dati applicativi. Per il CLDC, il numero di requisiti è minimizzato per massimizzare invece il numero di piattaforme sulle quali esso può essere implementato. Il CLDC assume che solo il dispositivo ha un sistema operativo che può eseguire e gestire la virtual machine.
Nonostante Java sia un ambiente di programmazione multithreaded, non è necessario per il sistema operativo avere a disposizione il concetto di thread o essere capace di schedulare più di un processo alla volta.
La CLDC consiste di una libreria cldc, di una Virtual Machine (KVM) e di un linguaggio di programmazione Java.

2.4.1 Linguaggio Java

Il linguaggio Java per il CLDC ha un limitato set delle caratteristiche invece presenti e disponibili sulla piattaforma J2SE. Alcuni punti che non vengono supportati sono i seguenti:
supporto alla virgola mobile;
metodo finalize();
gestione degli errori.

2.4.2 Caratteristiche della macchina virtuale

Le specifiche CLDC definiscono le caratteristiche che una VM deve avere per descrivere le Specifiche Java Language che non sono richieste. La Sun provvede una implementazione delle specifiche CLDC che è basata sulla KVM, che consiste in una VM capace di soddisfare i requisiti CLDC.

2.4.2.1 Limitazioni

Ci sono delle caratteristiche del linguaggio Java che non sono disponibili alle applicazioni CLDC.

Supporto al calcolo in virgola mobile

Da quando molti dei processori usati per la piattaforma CLDC non dispongono dell'hardware per la virgola mobile, alla virtual machine non è richiesto di supportare le operazioni in virgola mobile.
Questo significa che le operazioni qui sotto mostrate non vengono implementate.

Dadd	dload	dsub	fcmpl	fren	i2d
Daload	dload_x	d2f	fconst_0	freturn	i2f
dastore	dmul	d2i	fconst_1	fstore	l2d
dcmpg	dneg	d2l	fdiv	fstore_x	l2f
dcmpl	dren	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	

Figura 4: codici non implementati dalla VM CLDC

Questo conduce ad avere le seguenti restrizioni:
° le variabili del tipo float e double e gli array di questi tipi non possono essere dichiarati o usati.

° le costanti del tipo float o double del tipo (1.0, 2.0F) non possono essere usate.

Sun non fornisce una versione differente del Java compiler quando si sta sviluppando un'applicazione CLDC, dunque è possibile, usando un compiler J2SE, creare delle classi java che fanno uso dei floating point e dei suoi tipi; è da sottolineare il fatto che comunque queste classi verranno eliminate.

Reflection

Il package java.lang.reflect e tutti i tipi della java.lang.Class che sono connessi alla reflection non sono disponibli. Queste restrizioni sono applicate in particolare per salvare della memoria, ma viene anche salvata avendo determinato se il codice applicativo ha il privilegio di accedere a queste caratteristiche.

Object finalization

L'Object finalization causa una grande complessità nella VM per I pochi benefici. La finalization no viene implementata, e la classe CLDC java.lang.Object non dispone del metodo finalize().

Threads Group

La KVM implementa il multithreading ma non supporta i gruppi di thread. Le operazioni relative ai threads come la start o la stop invocate su un thread possono essere invocate soltanto sull'oggetto thread. Se si vogliono gestire delle operazioni su di un gruppo di thread, occorre utilizzare esplicitamente una collezione di oggetti al livello applicativo per gestire l'oggetto thread come gruppo.

JNI

La CLDC non prevede il JNI che permette ala codice nativo di essere chiamato dalle classi Java. Questa funzionalità è omessa perché richiede un intensivo uso della memoria nella sua implementazione e anche

perché si possono proteggere i dispositivi CLDC dai problemi che possono insorgere dalle manipolazioni di codice.

La virtual machine che supporta il CLDC è realizzata per identificare i classfiles non validi. Questa operazione è veramente costosa per un dispositivo CLDC in termini di memoria e consumo di potenza. Il CLDC definisce un differente meccanismo per implementare la verifica del classfile. L'operazione di pre-verifica prende luogo fuori dal dispositivo, per esempio, su una stazione server. Una volta che l'operazione di preverifica si è conclusa, il classfile viene scaricato sul dispositivo.

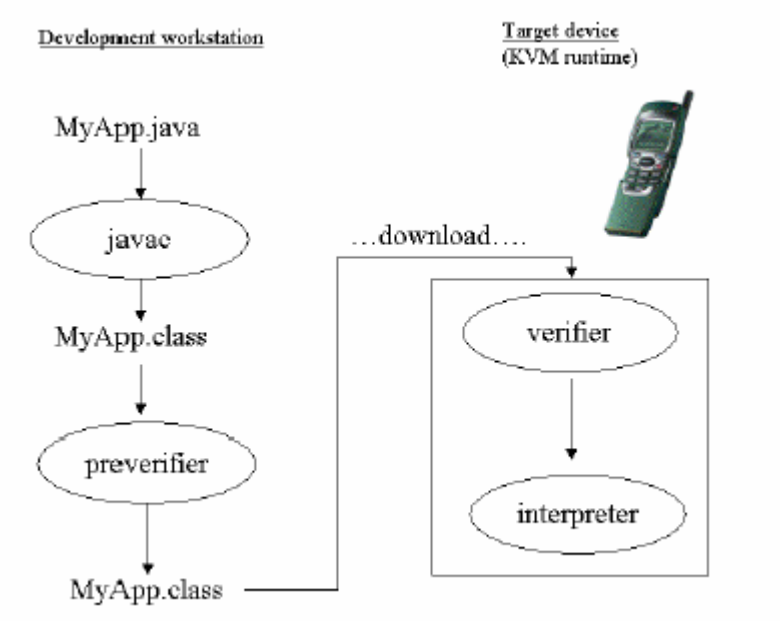

Figura 5: preverifica classfile in CLDC/KVM

2.4.2.2 Gestione della sicurezza

In J2SE, il modello di sicurezza è abbastanza potente da permettere di originare del codice da diverse sorgenti che hanno molti livelli di privilegi e diversi modi per accedere alle risorse del sistema. Le applicazioni installate su una macchina utente, di default, hanno degli accessi non ristretti. Un applet scaricata da un server web, comunque, opera in un ambiente ristretto che non permette di accedere alle risorse locali,come il filestore di un utente. La sicurezza permette dei privilegi che devono essere assegnati o negati ad una applicazione o ad una applet basandosi sulla veridicità dell'utente d'origine. Il codice deve essere testato con un certificato che garantisce la giusta provenienza. Può essere anche crittografato in modo che il destinatario possa essere sicuro del fatto che il codice non è stato soggetto a delle modifiche durante il trasferimento dalla sorgente. Una virtual machine CLDC può essere usata in un dispositivo che non necessita l'installazione di codice da parte dell'utente e questo ha molto meno bisogno di garanzie di sicurezza. Potrebbe essere usato in un telefono cellulare connesso ad una rete che permette ad una applicazione di essere scaricata; la rete dovrebbe essere soggetta allo stesso tipo di sicurezza che viene applicata ad un applet J2SE.
Sarebbe utile poter disporre di livello intermediari di sicurezza per quel codice che deve essere testato. Sfortunatamente, questo non è applicabile nei casi generici, perché la memoria e la potenza richiesti per implementare i modelli di sicurezza del J2SE, verificare la firma crittogafica e i certificati, sono per dispositivi risponedenti alle specifiche CLDC. La virtual machine CLDC fa girare le applicazioni in un ambiente noto come "sandbox" che assicura la protezione del dispositivo sul quale sta eseguendo.
Di default, la virtual machine J2SE esegue il byte-code su tutte le classi caricate da una sorgente esterna ma non su quelle caricate dal filesystem. In un ambiente mobile, è generalmente opportuni applicare queste verifiche a tutti i codici applicativi. Comunque, l'algoritmo necessario alla verifica richiede abbastanza memoria.

2.4.3 Le classi CLDC

Il CLDC indica un range di piattaforme che non hanno sufficiente memoria per supportare tutti i package e le classi previste dal J2SE.
Dato che il CLDC è una configurazione, non può avere delle caratteristiche opzionali. La libreria del CLDC è composta da un

package contenente le funzionalità specifiche del J2ME (chiamata javax.microedition.io).
Le librerie contenute in CLDC sono in parte derivate dalla piattaforma J2SE e in parte appositamente scritte per la specifica gestione dei dispostivi mobili.
Tutte le configurazioni J2ME e i profili includono package o classi della J2SE. Quando la J2ME incorpora delle interfacce software delle J2SE, deve seguire alcune regole:
° il nome dei package o delle classi deve essere lo stesso, se possibile;
° la semantica delle classi e dei metodi usati nella J2ME devono essere identici a quelli avente lo stesso nome nella J2SE;
° non è possibile aggiungere campi pubblici o protetti o dei metodi a classi che sono condivise da entrambe le piattaforme.

Alle configurazioni e ai profili J2ME non è permesso aggiungere delle funzionalità extra ai pacchetti e alle classi che condivide con la J2SE, dunque la compatibilità dalla J2ME alla J2SE è preservata.

Le classi derivate dalla piattaforma standard appartengono ai seguenti package:

• *java.io:* il CLDC provvede solo un limitato set del package java.io J2SE. Gli stream di input e output che possono essere utilizzati per una connessione verso un source o verso un sink sono il ByteArrayInputStream e il ByteArrayOutputStream. Questi stream possono essere usati per leggere o scrivere da/su un array di byte, o, possono essere wrappati con un DataInputStream o un DataOutputStream. L'accesso a tutte le sorgenti di dati è ottenuto mediante l'implementazione di un InputStream o di un OutputStream e questi sono ottenuti mediante la chiamata a dei metodi di altre classi. L'esempio più importante può essere quello dei metodi openInputStream() e openOutputStream() dell'interfaccia StreamConnection. Il package java.io fornisce supporto anche per i caratteri di input e output incapsulando dei byte stream con un InputStreamReader o un OutputStreamWriter. Comunque, le classi come FileReader e StringWriter non fanno parte delle specifiche CLDC.

• *java.lang*: include tutte le classi tipiche del linguaggio Java incluse nella Java 2 Standard Edition. In questo package troviamo tutte le classi per i tipi di dato (Boolean, Byte, Character, Integer, Long, Short, String, StringBuffer) oltre alle classi basi di sistema e per i Thread (Math, Object, Runtime, System, Thread, Throwable).
Da quando il package java.util non include la classe Properties, la classe system non include il metodo getProperty(), e dunque non è possibile recuperare una lista di tutele proprietà disponibili. Un dispositivo che supporta uno o più profili J2ME deve includerli nella proprietà microedition.profiles, e i profili tipicamente definiscono le loro proprietà in aggiunta a quelle disponibili.

• *java.util*: include tutte le classi di utility incluse nella Java 2 Standard Edition. In questo package troviamo le classi che un programmatore java è abituato ad utilizzare (Calendar, Date, Hashtable, Random, Stack, Timer, TimerTask, TimeZone , Vector).
Il package java.util contiene la collection class e delle classi relative alle date e alla gestione del tempo.

• *javax.microedition.io:* questo package contiene una collezione di interfacce che definiscono il Generic Connection Framework (GCF). Questo framework viene usato dai profili basati sul CLDC per prevedere un meccanismo di accesso alle risorse di rete ma anche ad altre risorse che possono essere chiamate o che possono inviare e ricevere dati mediante un InputStream e un Outputstream rispettivamente.
Un tipico esempio di queste risorse è una pagina HTML o una servlet Java, che possono essere identificate da un Uniform Resource Locator (URL). Nonostante le specifiche CLDC definiscono le interfacce e i metodi del framework e suggeriscono come sia possibile permettere alle applicazioni aprire delle connessioni ai vari tipi di risorse, le specifiche non richiedono alcuna attuale implementazione. Comunque, specificando i comuni metodi necessari per aprire, chiudere, e recuperare dati da queste risorse, il framework rende più facile agli sviluppatori la scrittura di applicazioni che possono connettersi a sorgenti dati usando diversi meccanismi di comunicazione, come socket, datagrammi, o http.

E’ possibile mettere in evidenza la relazione che esiste tra le librerie della J2SE e quelle del CLDC.
Le librerie del CLDC sono divise in due categorie: classi che sono un subset delle librerie J2SE e classi che sono specifiche del CLDC ma che possono essere mappate sul J2SE.

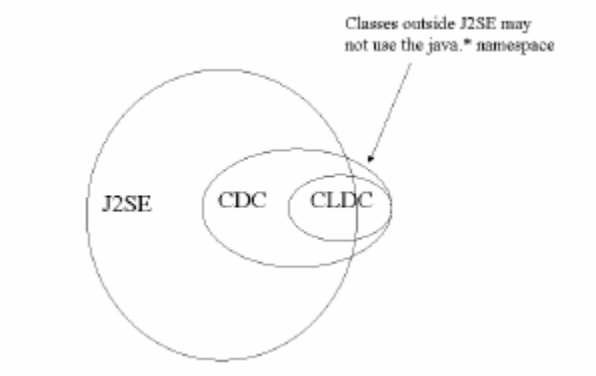

Figura 6: relazione tra la configuarazione J2ME e quella J2SE

I dispositivi mobili possono differire enormemente per quanto riguarda il supporto o l’assenza del filesystem e delle reti. Il *Generic Connection Framework (GCF)* è costituito da un gruppo di classi e di interfacce ideate per facilitare l’accesso alla memoria e ai sistemi di rete senza specificare i requisiti hardware e software. Nonostante che i package java.io e java.net standard contengano già molte funzioni complete per la risoluzione di questi problemi, non è possibile utilizzarli poiché richiedono circa 250 kilobyte e tale carico è impossibile da sopportare per molti dispositivi mobili. Al contrario, GFC è stato pensato per supportare molti apparecchi mobili e protocolli, sia già esistenti sia ancora da sviluppare.

2.4.4 Generic Connection Framework

GFC è stato sviluppato con lo scopo di estrapolare dai package java.io e java.net della Standard Edition le funzionalità principali e adattarle agli information device. È stato mantenuto un elevato grado di estensibilità e

flessibilità per garantire il supporto delle differenti forme di comunicazione e di nuovi protocolli.
GFC è implementato in un insieme di interfacce che rappresentano vari livelli di astrazione di metodologie di connessione. La decisione di non implementare direttamente, a livello di configurazione, i vari protocolli, ma di lasciare questo compito al livello applicazione (o profilo), rientra nella logica di un approccio generico al problema. In questo modo la configurazione fornisce gli "strumenti generici di base" permettendo il supporto di molti protocolli e l'utilizzo sui più disparati dispositivi (ad esempio una connessione di tipo datagram può avvenire attraverso l'IP ma anche tramite beaming su porta infrarossi).

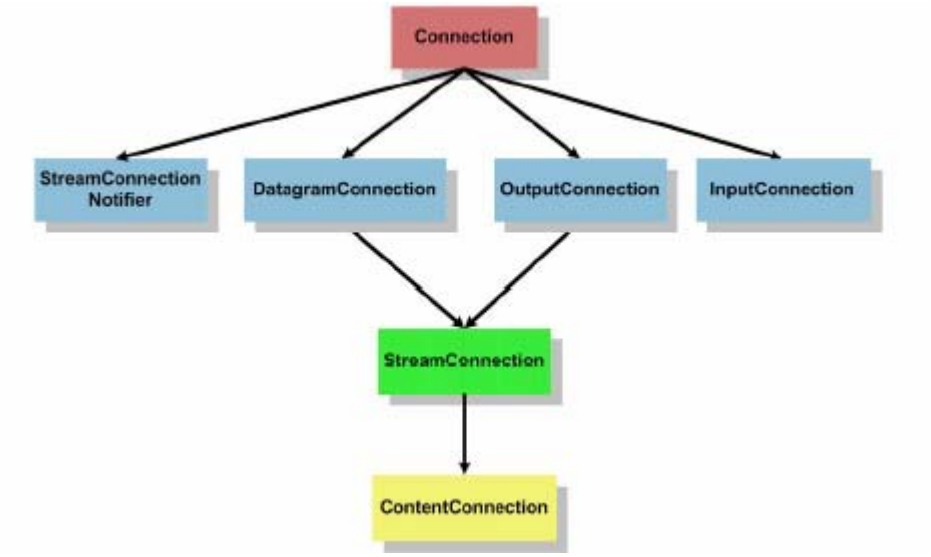

Figura 7: gerarchia delle interfacce di connessione

Alla radice della gerarchia, l'interfaccia Connection, rappresenta una generica connessione che può essere aperta e chiusa attraverso i metodi open()e close(). InputConnection e OutputConnection rappresentano un dispositivo dal quale i dati possono essere rispettivamente letti o scritti attraverso gli opportuni stream, StreamConnection è la combinazione di Input/Output-Connection e costituisce normalmente il punto di partenza per le classi che implementano le interfacce di comunicazione. La sua estensione ContentConnection dispone di alcuni metodi (getEncoding,

getType, getLength) che consentono di ottenere informazioni sui dati trasmessi.
Come è possibile vedere in figura sono presenti altre due interfacce discendenti direttamente da Connection: StreamConnectioNotifier e DatagramConnection. La prima rappresenta una connessione server-side di tipo socket, il metodo acceptAndOpen() crea una connessione con il client e ritorna una StreamConnection (questo tipo di connessione è simile ai server socket della Standard Edition). DatagramConnection può essere utilizzata per implementare connessioni di tipo datagram. Una connessione di tipo datagram può essere aperta in modalità "server" oppure "client": nel primo caso il dispositivo resta nell'attesa di ricevere datagram, nel secondo caso è lui ad iniziare la trasmissione.
Nel package javax.microedition.io, che è il "contenitore" di tutti gli strumenti di connessione troviamo, oltre alle già citate interfacce, contiene un unica classe "concreta": Connector. Tale classe possiede sette metodi statici che permettono l'apertura effettiva della connessione sulla quale sarà poi possibile collegare gli stream di I/O, per le operazioni di lettura e/o scrittura. Indipendentemente dal tipo di connessione che si intende utilizzare occorrerà richiamare il metodo (statico) open per poterla utilizzare in lettura e/o scrittura. Il metodo open prevede tre firme differenti:
public static Connection open(String connectionString)
public static Connection open(String connectionString, int mode)
public static Connection open(String connectionString, int mode, boolean timeouts)
Il parametro connectionString rappresenta la stringa di connessione ed ha un formato particolare che verrà illustrato tra breve, il valore intero mode indica la modalità di apertura della connessione e può assumere uno dei seguenti valori interi (variabili intere statiche della classe Connector):
• *READ:* apertura in sola lettura;
• *WRITE:* apertura in sola scrittura;
• *READ_WRITE:* apertura sia in lettura che in scrittura (valore di default);
Il booleano timeout, il cui valore di default è false, se impostato a true indica che il metodo può generare eccezioni dovute al timeout del tipo InterruptedIOException.
Per quanto riguarda la connectionString, essa ha una generica

forma del tipo: *[protocol] : [target] [parameters]*

• *protocol:* indica il tipo di connessione che si vuole attivare (ad esempio: I/O su file, porta seriale, datagram, socket, Http).
• *target:* dipende da protocol e può essere il nome di un file, di una porta seriale, il nome di una porta di comunicazione o il nome di un host.
• *parameter:* sono parametri aggiuntivi (opzionali) che forniscono informazioni funzionali per un dato tipo di connessione. Ad esempio la velocità, i bit di start e stop e la parità di una connessione seriale.
Le operazioni di apertura di una connessione possono generare eccezioni, in particolare se la connectionString passata ad un metodo di apertura" non è ben formata si avrà una IllegalArgumentException, mentre se essa indica un tipo di protocollo non supportato verrà generata una ConnectionNotFoundException. Una eccezione di tipo IOException, infine, sta ad indicare che si è verificato un qualche errore di I/O.

2.5 J2ME e i protocolli di comunicazione

La programmazione di rete gioca un ruolo importante nello sviluppo delle applicazioni wireless ed offre il vantaggio della connettività che questi dispositivi devono offrire. E'stato introdotto il concetto di Generic Connection Framework ed il fatto che le applicazioni MIDlet fanno uso di diversi tipi di comunicazione disponibili nel framework, come le socket e la connessione http.

Il Generic Connection Framework supporta le seguenti basi di comunicazione. Tutte le comunicazioni sono create mediante il metodo Connector.open()

HTTP:
Connector.open("http://www.webyu.com");
Sockets:
Connector.open("socket://localhost:80");

2.5.1 Protocollo http

Il protocollo **http** è basato sul TCP, e definisce un metodo di interazione cliente-servitore che ottimizza l'affidabilità delle comunicazioni. E'

riferito al livello *applicazione* dello standard OSI e rappresenta un protocollo stateless e leggero. Stateless significa che il server non ha memoria delle connessioni effettuate e quindi tratta le connessioni tutte allo stesso modo.
Leggero significa che il client si connette al server solo per il tempo necessario per trasmettere la risorsa.

2.5.2 HttpConnection

Il protocollo http è basato su una HttpConnection che permette una comunicazione con un web server. L'Httpconnection definisce dei metodi che permettono di fare la programmazione di rete basata su http molto più semplice. Ad esempio, l'HttpConnection provvede dei metodi che permettono agli sviluppatori di ottenere le informazioni dell' header http facilmente.
Usare un'HttpConnection come comunicazione di rete in un'applicazione offre molti vantaggi:
1. Non tutti i dispositivi MIDP supportano la comunicazione socket o datagram

2. La comunicazione socket o datagram sono molto dipendenti dalla rete. Alcune reti possono implementare solo alcuni tipi di comunicazione e no altri. Questo tipo di limitazione rende le applicazioni wireless molto meno portabili.

3. Il supporto del protocollo http nei dispositivi MIDP dà alle applicazioni wireless un protocollo di alto livello e indipendente dalla rete con cui poter lavorare. Le applicazioni wireless sviluppate usando un'HttpConnection sono portabili attraverso diversi tipi di rete.

4. Nella richiesta http possono essere incapsulati molto più facilmente diversi tipi di dati

L'interfaccia http supporta un subset del protocollo http 1.1.
Alcuni metodi dell'HttpConnection sono:

String getHeaderField(String name)

Questo metodo ritorna il valore del nome dell'intestazione.

String getHost()

Questo metodo ritorna delle informazioni sull'URL.

int getPort()

Questo metodo ritorna il numero di porta dell'URL. Come valore di default ritorna il valore 80 se non è stato passato nessun valore nella stringa *connector.open().*

Il *connector.open()* è un metodo che viene utilizzato per aprire una connessione e che riceve dei parametri del tipo: *{protocol}://[{host}]:{port};* cioè occorre specificare il tipo di protocollo utilizzato nell'aprire una connessione, l'host e il numero di porta.

2.5.2.1 Stati dell'http

Esistono due possibili stati per una connessione http: *Setup e Connected.* Nello stato di *Setup*, la connessione non è ancora stata aperta verso il server. Nello stato di *Connected*, invece, la connessione è stata avanzata al server, i parametri sono stati inviati e la risposta viene attesa sempre in tal stato.
La transizione dallo stato di Setup allo stato Connected è causata da alcuni metodi che richiedono dei dati che devono essere inviati o ricevuti dal server.

2.5.2.2 Metodi di richiesta dell'http

La connessione http dispone di tre tipi di richiesta da esser inviati ad un web server: GET, POST, HEAD.

Il metodo GET è usato dai programmi per ottenere i contenuti di un documento web da uno specifico url. La risposta del server web consiste

di un header http contente delle informazioni sul documento web, informazioni tipo MIME sul contenuto dei dati.

Il metodo POST è spesso usato dai programmi per inviare delle informazioni all'url su programmi CGI. Entrambi i metodi post e get possono essere usati per inviare dei dati ad un programma CGI;la differenza è che il metodo post manda i dati mediante uno stream mentre il metodo get manda i dati mediante variabili d'ambiente presenti nella stringa di query.

2.5.3 SocketConnection

La socket è un end-point di una comunicazione a due vie tra programmi che funzionano su una rete. Una connessione socket è la comunicazione di basso-livello realizzabile tra un dispositivo wireless e un server remoto oppure tra due dispositivi wireless.
La capacità di una comunicazione socket prevista con alcuni dispositivi mobili in J2ME permette una varietà di applicazioni client/server.
L'utilizzo delle socket dà agli utilizzatori la flessibilità di sviluppare molti tipi di applicazioni di rete per dispositivi wireless. Comunque, non tutte le realizzazioni wireless supportano la comunicazione socket nei dispositivi MIDP, il che significa che le applicazioni wireless sviluppate e facenti uso delle socket potrebbero essere limitate ad alcuni dispositivi e potrebbero essere molto meno portabili attraverso differenti tipi di reti wireless.

Per poter usare una socket, il mittente e il destinatario devono stabilire una connessione socket prima di poter comunicare. Una volta effettuato questo, l'uno rimarrà in attesa di richieste di connessione e l'altro chiederà una connessione. Quando le socket saranno connesse, sarà possibile trasmettere i dati in entrambe le direzioni.
Per ricevere i dati da un server remoto, viene stabilito un *InputConnection* e dalle connessione viene ricevuto un *InputStream*.
Per inviare i dati ad un server, viene invece stabilito un *OutputConnection* e dalla connessione deve essere ottenuto un *OutputStream.*

In J2ME, vengono definiti tre tipi di connessione per gestire gli output/input stream: *InputConnection, OutputConnection, e StreamConnection.*
Come indicato, un InputConnection definisce la capacità per uno stream di ricevere i dati, un OutputConnection definisce la capacità per uno stream di inviare i dati. Lo StreamConnection definisce invece la capacità per entrambi gli stream di output ed input.

La programmazione di rete facente uso delle socket è veramente importante nelle JME. Il processo avviene come qui descritto:
1. Una connessione socket viene aperta con un server remoto o con un qualsiasi dispositivo wireless facendo uso del metodo connector.open().

2. Un InputStream o un OutputStream viene creato dalla connessione socket per inviare o ricevere i pacchetti dati.

3. I dati possono essere inviati o ricevuti dal server con una connessione socket eseguendo delle operazioni di scrittura o lettura sull'oggetto OutputStream o InputStream.

4. La connessione socket e gli stream di input ed output devono essere chiusi prima di uscire dal programma.

2.6 Mobile Information Device Profile

Il CLDC fornisce le basi per lanciare Java sui dispositivi che hanno risorse insufficienti per supportare una virtual machine insieme ad una versione completa dei package della J2SE.
Il *Mobile Information Device Profile o* MIDP è un profilo per essere usato sui dispositivi che hanno un'interfaccia utente limitata nella forma di un piccolo video con poche funzionalità di input.
MIDP è una versione della piattaforma Java basata sul CLDC e sulla KVM che è indirizzata per dispositivi di piccole capacità come i cellulari. E' disponibile anche per dispositivi come i PDAs e i palmOS versione 3.5.
Il software che implementa MIDP esegue nella KVM fornita del CLDC e provvede a servizi addizionali per beneficiare il codice scritto usando le api del MIDP.

Le applicazioni MIDP sono chiamate MIDlets. Una MIDlet può utilizzare le facilità introdotte dal MIDP e le api inerenti. Le Midlet non accedono al sistema operativo della piattaforma e non possono così farlo senza diventare nonportabili. Dato che la KVM non supporta JNI, l'unica via per un'applicazione MIDlet di accedere alla piattaforma nativa è di collegare il codice nativo in un aversione castomizzata della virtual machine.

Sun provvede ad un'implementazione del MIDP che può essre usta su Windows, ed è il Wireless Toolkit, che contiene la versione del MIDP per Windows, Solaris e Linux.

La sezione denominata *"OEM Code" (Original Equipment Manifacturer)* racchiude una serie di servizi derivati da CLDC e MIDP che sfruttano le caratteristiche del sistema operativo e della macchina host. Per questo motivo anche tale supporto deve essere fornito dal produttore dell'apparecchiatura. Sfruttando i servizi di tale sezione si otterranno delle applicazioni non portabili su altri dispositivi.

2.6.1 Requisiti

Il MIDP è inteso per dispositivi che hanno delle capacità limitate e stiamo parlando di dispositivi come i cellulari e i palmari. Adesso vengono riportati i requisiti hardware minimi richiesti.

2.6.1.1 Memoria

Le specifiche MIDP richiedono almeno 128 KB di RAM per poter gestire l'applicazione stessa. In aggiunta a questa ne occorrono 32 KB per l'heap Java. Un heap di 32 KB è veramente limitato e richiede che lo sviluppatore eserciti molta cura quando alloca gli oggetti e richiede anche che faccia tutti i possibili passi per evitare di tenere i riferimenti agli oggetti più del necessario, in modo da permettere al garbage collector di liberare lo spazio il più presto possibile. I dispositivi MIDlet richiedono inoltre 8 KB di memoria non volatile da essere usata come storage persistente di modo che la MIDlet possa salvare le informazioni anche quando il dispositivo viene spento.

2.6.1.2 Display

I dispositivi MIDP sono caratterizzati da display di piccole dimensioni. Le specifiche richiedono che lo schermo abbia almeno 96 pixel di lunghezza e 54 pixel di altezza. Lo schermo deve avere almeno due colori e vi sono dei dispositivi che comunque no riescono ad andare oltre queste caratteristiche.

2.6.1.3 Input Device

Ci sono sette tipi di differenti dispositivi di input che possono esser trovato in una piattaforma MIDP. Uno dei dispositivi più sofisticati come la RIM, ha una tastiera alfanumerica.
Questa si ritrova anche dei dispositivi cellulari e può essere realizzata in diverse maniere.

Figura 8: tipica tastiera di un cellulare

2.6.1.4 Connectivity

I dispositivi mobili hanno alcune modalità d'accesso alla rete, può essere la connessione wireless in un dispositivo cellulare, o un modem per un PDA.
MIPD non assume che i dispositivi siano necessariamente connessi alla rete o che la rete supporti il TCP/IP. Richiede invece che il dispositivo supporti al connessione http, direttamente con un protocollo Internet o utilizzando una connessione wireless mediante un gateway WAP.

2.7 Requisiti Software

L'implementazione rende le assunzioni sulle capacità offerte dal software sul quale esso è presente.

Il sistema operativo deve provvedere un ambiente d'esecuzione nela quale la JVM può eseguire. Dato che il CLDC supporta le capacità di threading del J2SE, la piattaforma supporta il multithreading e, se è così, la KVM può fare direttamente uso di essa.
Su alcune piattaforme un socket-level è disponibile, sul quale un supporto http midp può essere implementato.
Il software deve essere capace di trattare degli eventi quando un tasto viene premuto e rilasciato.
La piattaforma deve provvedere alcune forme di memorizzazione persistente che non perde il suo stato quando il dispositivo viene spento. Il MIDP prevede una modalità di accesso a questa forma di memorizzazione che prevede l'uso di ***record*** e richiede che il software gestisca alcuni tipi di interfacce.

2.8 MIDlets

Le applicazioni Java che eseguono su dispositivi MIDP sono conosciute come MIDlet. Una MIDlet consiste di almeno una classe java che deve derivare dalla classe astratta javax.microedition.midlet.MIDlet. Una MIDlet esegue in un ambiente di esecuzione dentro la VM che provvede un ciclo di vita controllato dai metodi della classe *MIDlet* che ogni MIDlet deve implementare. Una MIDlet può anche usare dei metodi della classe MIDlet per ottenere dei servizi dal suo ambiente, e dve udìasre solo le api che sono definite nelle specifiche MIDP.
Un gruppo di MIDlet costituisce un Midlet suite. Le MIDlet presenti in una suite condividono risorse statiche e dinamiche del loro ambiente:
° in runtime, se il dispositivo supporta l'esecuzione concorrente di più di una MIDlet, tutte le MIDlet attive in una suite eseguono sulla stessa VM Java. Tutte le MIDlet presenti in una suite condividono le istanze di tutte le classi java e tutte le risorse caricate sulla VM.

°la memorizzazione persistente sui dispositivi è gestita a livello di suite delle MIDlet. Per una MIDlet non è comunque possibile ottenere l'accesso alla memorizzazione persistente gestita da un'altra MIDlet.

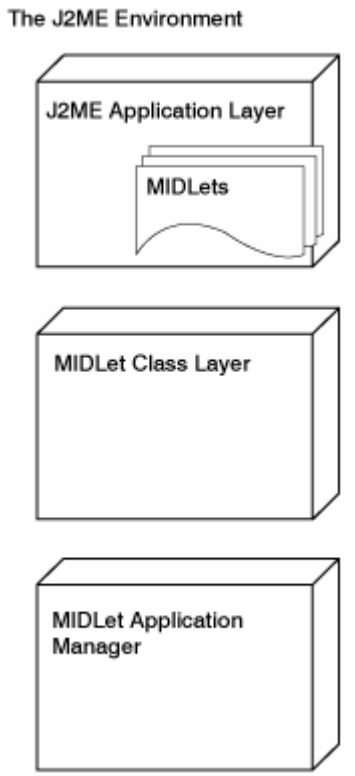

Figura 9: ambiente J2ME

2.8.1 Sicurezza MIDlet

Il modello di sicurezza utilizzato nel J2SE è potente e flessibile, ma è costoso in termini di risorse di memoria. Un possessore di dispositivo mobile deve stare attento quando installa una MIDlet perché occorre prima testare e accettare software solo da risorse testate. Sfortunatamente, quando è il momento, non c'è modo per l'utente di essere sicuro che il codice non sia stato interferito; il meccanismo di autenticazione che è previsto per la piattaforma J2SE non è uno standard per la piattaforma MIDP. La versione sicura del protocollo http, che

aiuterà ad alleviare questo problema, è da prendere in considerazione per una futura versione del MIDP. Non ci sono api MIDlet che permettono l'accesso a delle informazioni già presenti sul dispositivo.

2.8.2 MIDlet Packaging

Per rendere l'applicazione scaricabile ed istallabile è necessario che tutti i file necessari al MIDlet vengano inseriti in un file JAR. I file JAR (*Java Archive*) costituiscono una suite MIDlet. Questi archivi contengono tutti i file sorgenti delle varie applicazioni che devono collaborare in una suite oltre ai file di supporto come immagini, suoni e dati.
Un JAR può contenere più di un a MIDlet; tutte le MIDlet che allora fanno parte di una suite devono essere presenti in un JAR.
Una MIDlet è caratterizzata da alcuni attributi come: Name, Version, Description, Icon. Questi attributi sono utili all'utente per capire come la MIDlet deve essere installata.
Un particolare file contenuto nei JAR è il *manifest.mf*: è un file di testo il cui compito è descrivere il contenuto dell'archivio attraverso una serie di attributi. Questi attributi riportano informazioni di tipo generale sulla suite come il produttore, la versione, una breve descrizione generale ed altri elementi di carattere tecnico come configurazione, profilo e riferimenti ad ulteriori risorse.
Oltre agli archivi JAR è possibile la presenza di file JAD (*Java Application Descriptor*). Sono file di testo che contengono una serie di attributi di sistema e personalizzati: i primi vengono utilizzati dal gestore delle applicazioni per reperire il file JAR, i secondi possono essere aggiunti dal programmatore e recuperati nei vari MIDlet tramite opportuni metodi.
Come dice il nome stesso, i file JAD hanno lo scopo di fornire informazioni sui MIDlet che fanno parte della suite, tali informazioni sono necessarie al *Application Manager Software (AMS)* per verificare se la suite di MIDlets è in grado di operare sul dispositivo. Sulla base degli attributi contenuti nel file JAD, l'AMS è in grado di valutare la possibilità o meno di procedere all'installazione dell'applicativo. Ad esempio, l'AMS verifica se esiste spazio di memoria sufficiente per il file JAR, valuta la corrispondenza delle versioni di configurazione e

profilo, si assicura che nel dispositivo non sia presente la stessa versione dell'applicativo che si intende installare.

2.8.3 Ciclo di vita di un MIDlet

Tutti i MIDlet derivano dalla classe astratta javax.microedition.midlet.MIDlet, la quale contiene i metodi che la piattaforma MIDP mette a disposizione per il controllo del ciclo di vita di un MIDlet. Un MIDlet è un'applicazione costruita sulla classe javax.microedition.midlet.MIDlet. Il gestore delle applicazioni comunica con un MIDlet tramite i metodi messi a disposizione da queste classe. La comunicazione è bidirezionale. Per esempio: come il gestore delle applicazioni può mettere in pausa un MIDlet, così un MIDlet può richiedere di essere messo in pausa. Il ciclo di vita è composto da tre stati principali: *ACTIVE, PAUSED, DESTROYED.*

Dopo l'invocazione del costruttore e appena viene caricato, il MIDlet viene messo nello stato *Paused.* Se durante l'esecuzione del costruttore si verificano problemi viene lanciata un'eccezione e il MIDlet si porta nello stato *Destroyed*, altrimenti si mette in pausa ed è pronto per essere messo in esecuzione dal gestore delle applicazioni.
L'applicazione viene attivata lanciando il metodo astratto startApp(). Se non si riscontrano problemi, il metodo fa sì che l'applicazione MIDlet si attivi e cominci ad operare, altrimenti il gestore delle applicazioni risponde con determinate operazioni a determinati errori:
• se sulla piattaforma si riscontra un errore relativo ad una condizione momentanea o il codice del metodo startApp() solleva una qualsiasi eccezione che non viene catturata, il gestore delle
applicazioni solleva l'eccezione MIDletStateChangeException ed il MIDlet viene riportato nello stato di pausa;
• se il gestore delle applicazioni rileva un errore che non è possibile recuperare il MIDlet viene distrutto con il metodo destroyApp();
A questo punto il MIDlet è in esecuzione e può passare tra gli stati
Paused e Active senza alcun limite.
Con il metodo destroyApp() il MIDlet libera tutte le risorse che ha allocato durante l'esecuzione e termina ogni thread in esecuzione in background. Se l'argomento passato è *true* significa che il MIDlet si

deve portare in ogni modo nello stato Destroyed, se è *false* si dà la possibilità all'applicazione di sollevare l'eccezione MIDletStateChangeException e far sì che rimanga nello stato Active. Quando il MIDlet necessita di auto-terminarsi, ad esempio se l'utente ha deciso di uscire dall'applicazione, deve notificare al gestore delle applicazioni questa intenzione attraverso il notifyDestroyed() e liberare tutte le risorse impegnate. Se un MIDlet vuole essere messo in pausa, notifyPaused() invia la richiesta al gestore delle applicazioni, mentre nel caso in cui voglia uscire da tale stato presenta la richiesta con il metodo resumeRequest().

2.9 MMAPI

2.9.1 Il package MMAPI

L'MMAPI è un package Java che permette ad un dispositivo di accedere alle funzionalità multimediali time-based, come clip audio, sequenze MIDI, animazioni. I multimedia time-based giocano un importante ruolo in molte applicazioni; una delle più importanti applicazioni è sui servizi on-line. Con le MMAPI, i servizi che possono essere resi disponibili ad un utente sono:
° servizi di news arricchiti
con video e audio clip contenenti dei commenti;
° servizi di messaggistica, dove immagini, video e audio possono essere trasmessi con il messaggio di testo;
° giochi interattivi on-line che integrano anche capacità video e di suoni;
° cattura di immagini su display;
° streaming di immagini, video e suoni;
° servizi di sicurezza come i sistemi di emergenza;
° comunicazioni interne;
Per i clienti, disporre di questi servizi su un dispositivo mobile enfatizza l'attività che possono ottenere sul loro display. Gli utenti vedono sempre più i loro telefoni cellulari come degli strumenti da personalizzare per i loro affari e per gli usi d'intrattenimento.

Mentre i dispositivi wireless hanno dei problemi di memoria limitata, le loro capacità di processing invece stanno divenendo sempre più potenti. Molti PDA attuali sono potenti come macchine desktop di alcuni anni

fa. Sta divenendo sempre più possibile realizzare servizi multimediali fruibili da questi dispositivi. I vantaggi di Java 2 Micro Edition (J2ME) per questo nuovo mondo delle piccole applicazioni, conosciute come MIDlets, è che esse sono così piccole (10-30k) che impiegano poco tempo per effettuare il download e usano solo una piccola frazione della memoria disponibile.
Per supportare i servizi multimedia che le MIDlets possono eseguire, sono state sviluppate delle API specifiche chiamate MMAPI.
L'MMAPI è un package opzionale nella piattaforma J2ME che provvede un supporto ai multimedia time-based sui dispositivi wireless. Tale package e' stato pensato per venire incontro alle stringenti caratteristiche dei dispositivi portatili, ma anche ai problemi inerenti ad una connessione wireless: larghezza di banda limitata, traffico burst, connessione lenta, piccole risoluzioni di schermo. Queste API permettono un buon accesso e un controllo di un multimedia time-based, come semplici sequenza audio, video e MIDI.

2.9.2 Caratteristiche delle MMAPI

Le MMAPI sono realizzate per eseguire su delle Java virtual machine con le funzionalità del CLDC.
I concetti fondamentali delle MMAPI sono il Player e il DataSource. Il Player accetta i dati, li decodifica e fa un rendering sul display del dispositivo. Il contenuto del media può esser video o audio. I dettagli riguardanti quali codifiche sono supportate sono lasciati al profilo (come il MIDP).
Il DataSource incapsula la gestione del protocollo. Esso tiene conto di come i dati vengono letti dalla sorgente e gestisce anche i dettagli di connessione: per esempio, se il protocollo utilizzato è l'http o l'rtp.
Il player e il DataSource sono realizzati per essere flessibili. E' possibile creare un DataSource per le applicazioni che usano il loro specifico protocoll di input. Il Player può allora essere letto dal DataSource e così si può effettuare l'esecuzione del media. Per le applicazioni che eseguono su server J2SE o J2EE, che hanno molte più risorse, occorre utilizzare il JMF.

Le MMAPI possono essere parte di un'applicazione client-server: si può avere un'applicazione JMF su un server comunicante con un dispositivo client realizzato con le MMAPI.
Entrambi le API sono protocol-neutral, ciò significa che il canale di comunicazione può essere settato all'uso di un protocollo http, o rtp o un qualunque protocollo.
Dato che è impossibile predire il tipo o il dispositivo al quale le MMAPI sono affidate, le api non possono inviare quale tipo di formato viene supportato. Le MMAPI permettono di implementare solo il formato che un particolare dispositivo supporta. Per esempio, se si utilizzano le MMAPI su una macchina karaoke, occorre implementare la cattura dell'audio e la sua rappresentazione; non è necessario supportare un formato video o i controlli ad esso relativi.
In molti dispositivi, non c'è abbastanza potenza per realizzare delle funzionalità come la codifica dei media. Per esempio, i telefoni cellulari hanno una implementazione nativa dell'esecuzione audio o di una sequenza di toni. In questi casi, le MMAPI diventano un piccolo wrapper Java che permette alle applicazioni java di acceder ai servizi nativi.

Queste api sono state realizzate per essere agnostiche ai formati e ai protocolli. Per esempio, le api non necessitano di supporti per protocolli come http o rtp o formati come MP3, MIDI, MPEG-4. Le API contengono tutte le funzionalità che occorrono per supportare questi protocolli e molti altri.
Sottolineiamo il fatto che le MMAPI sono state realizzate per eseguire su uno stack CLDC/MIDP, che è inteso per piccoli dispositivi come i telefoni cellulari, dispositivi mobili, e altri. Possono anche eseguire su una configurazione CDC che supporta applicazioni java-language su dispositivi con poca memoria come i PDAs.

2.9.3 Concetti fondamentali delle MMAPI

Vediamo più nel dettaglio le astrazioni principali sulle quali si basano le MMAPI:

Player che accetta e decodifica i dati. Il Player è completamente neutrale ai tipi di dati che riceve. Dato ciò, non occorre un player particolare per dati video e uno per dati audio. Se i dati possono essere eseguiti dipende invece dai controlli che sono associati ad un player.

I tipi di media che possono essere rappresentati dipende dai *controlli* che vengono associati al player. Per essere riprodotti, ogni tipo di media richiede che vengano aggiunti uno o più controlli al Player. Per esempio, se si vuole riprodurre un dato audio, occorrerebbe aggiungere un controllo per il volume e un controllo per settare il momento in cui occorre fermare il player. Il controllo sul volume modifica il volume del Player; il controllo sul tempo di stop determina quando l'esecuzione del media termina.

Un *DataSource* provvede invece alla gestione del protocollo e ai metodi che controllano l'esecuzione del media e la sincronizzazione. Esso gestisce i dettagli sulla modalità di lettura dei dati dalla sorgente: un file, un server streaming, ecc.

Il *Manager* cattura ogni cosa nel momento della creazione del Player e associa un DataSource. Il Manager conduce le domande relative ai tipi di media supportati e ai protocolli.

I *Controlli* costituiscono le azioni che possono essere svolte su determinati tipi di contenuti multimediali e vengono resi disponibili al Player solo dopo che è stato risconosciuto il "content-type" della presentazione. Per ogni tipo di presentazione vi si possono associare solo determinati controlli.
Esempi di controlli applicabili ad un player sono:
VideoControl: è il controllo necessario alla realizzazione di player video; esso si occupa del rendering sui dispositivi di output.

2.9.3.1 Player

Il player è un oggetto *Mediahandler* per il controllo e il rendering del media. E' la classe principale per la gestione e il rendering e provvede un modello di alto livello del processo per usare il codice che lo separa

dalle considerazioni di basso livello come il formato delle tracce del media.
L'interfaccia del player estende quella del *Controller;* il player non solo provvede tutti i metodi per un controller, ma supporta anche cinque stati di un controller perché quest'ultimo è caratterizzato da un numero di stati superiore.
Le funzioni messe a disposizione da un player consentono di non interessarsi direttamente delle chiamate al codice nativo, di impegnare le risorse necessarie per la riproduzione e di effettuarne un eventuale rilascio, quando queste non siano più necessarie.
All'arrivo dei dati, si effettua l'inizializzazione del Player. Una volta inizializzato il player sarà possibile effettuare uno streaming dei dati multimediali.

2.9.3.1.1 Ciclo di vita di un Player

Il player prevede un ciclo di cinque stati: *UNREALIZED, REALIZED, PREFETCHED, STARTED, CLOSED.*

Quando un player viene realizzato per la prima volta, esso si trova nello stato Unrealized. La transizione dallo stato Unrealized allo stato **Realized** avviene quando il player effettua la comunicazione necessaria per allocare tutte le risorse di cui necessita per funzionare (come la comunicazione con un web server o un file system).
Il metodo *realize* permette ad un'applicazione di iniziare questo processo e di ottenere le informazioni richieste per acquisire la risorsa media. L'unico modo per ritornare allo stato unrealized è deallocare la risorsa, e questo può essere effettuato invocando il metodo *deallocate*. Normalmente, il player continua il suo corso passando allo stato Prefetched, e infine allo stato Starter.
Nello stato **Prefetched**, il player potrebbe aver bisogno di eseguire altri compiti, prima di eseguire la start. Per esempio, potrebbe dover acquisire delle risorse esclusive, i buffer per i media, ecc.
Nello stato **Started**, il player sta processando ed eseguendo i dati.
Un player si ferma quando raggiunge l'end-of-media o quando viene invocato il metodo *stop*. Quando accade ciò, il player passa dallo stato starter allo stato prefetched, ed è pronto per ripetere il ciclo.

Quando viene raggiunto lo stato **Closed**, il player rilascia tutte le risorse precedentemente acquisite e a questo punto non dovrebbe essere più utilizzato.

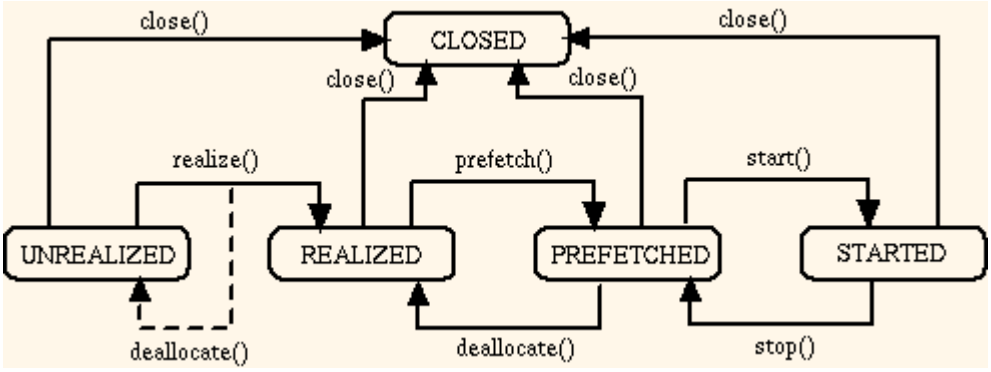

Figura 10: ciclo di vita di un player

2.9.3.1.2 Eventi Player

Gli eventi lanciano delle informazioni sui cambiamenti di stato del Player e su altre informazioni che provengono dai controlli sul Player.
Per ricevere eventi, un oggetto deve implementare l'interfaccia PlayerListener ed usare il metodo addPlayerListener per registrare il suo intersse a coglier gli eventi del Player. Gli eventi sono garantiti per essere colti nell'ordine che le azioni rappresentano l'avvenuto evento. Per esempio, se un Player si ferma dopo una start, l'evento Starter deve sempre precedere l'evento END_OF_MEDIA.
Un evento Error dever esser inviato nel momento in cui si verifica un errore irreversibile. Quando si verifica questo, il Player entra nello stato closet. Il meccanismo degli eventi è estendibile a diversi Player. Per gestire una lista di eventi scatenati da Player, occorre disporre di un'interfaccia PlayerListener.

2.9.3.2 Manager

Il Manager è il punto di accesso per ottenere le risorse dipendenti dal sistema come il Player per processare i media.

Il Player è un ggetto usato per controllare e rappresentare i media. Il Manager provvede gli accessi per un meccanismo specifico di implementazione per realizzare i Player.

Manager si occupa della gestione a "basso livello" dei player, nel senso che gestisce l'allocazione e la deallocazione di tutti i dispositivi (display, altoparlante, microfono, fotocamera, ecc.) di cui necessitano i player per il loro funzionamento e fornisce inoltre informazioni sull'hardware sottostante.
Attraverso la classe Manager si può ottenere un'istanza di Player utilizzando il metodo *createPlayer* disponibile in tre diverse forme:

public static Player createPlayer(InputStream is, String type)
public static Player createPlayer(String locator)
public static Player 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 *type* possono essere:
audio/x-wav : audio in formato Wave;
audio/basic : audio in formato AU;
audio/mpeg : audio in formato MP3 ;
audio/midi : audio in formato MIDI;
audio/x-tone-seq : sequenza di toni;
video/mpeg : video in formato mpeg;

Con la seconda forma, il player creato passando come parametro il media-locator. La stringa passata, locator appunto, non è una stringa arbitraria ma una URI e come tale ha un ben preciso formato e deve sottostare alle regole previste per le uri (schema, caratteri riservati, caratteri di escape, ecc).
L'ultima forma prevista, prevede il passaggio di un DataSource. Abbiamo detto che Manager costituisce il "gestore" del sistema sottostante ed in quanto tale è l'unico punto da cui è possibile ottenere informazioni riguardanti l'ambiente nel quale le MMA operano. Il metodo getSupportedContentTypes(String protocol) ritorna una lista di tutti i content-type supportati dal protocollo passato come parametro (protocol).
Il metodo public String[] getSupportedProtocol (String contentType) ritorna, invece, una lista dei protocolli previsti per un dato content-type

passato come parametro. Se viene passato null, la lista contiene l'elenco di tutti i protocolli supportati dall'implementazione.

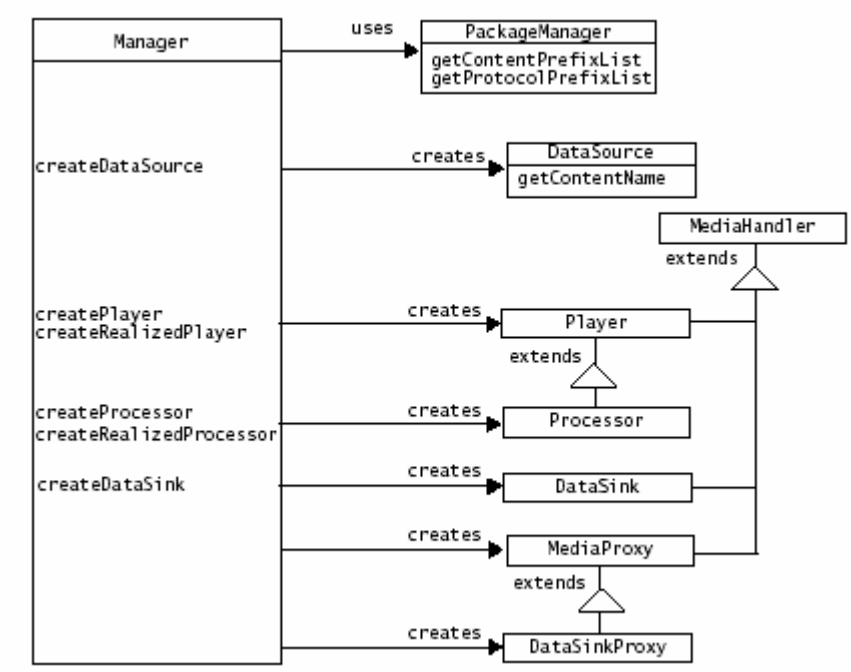

Figura 11: gestione media

2.9.3.3 DataSource

I media processati possono essere file video sul file system locale, traccie audio su un cd, streaming attraverso internet, o un range di altre possibilità. Tutti questi sono esempi di sorgenti media, e le api provvedono un modo per descriverli e gestirli mediante la classe che è il DataSource.
La classe DataSource fornisce un astratto e semplice modello dei media che possono essere trattati come parte di un processo: la creazione di un player, mediante la classe Manager, coinvolge prima la creazione di un DataSource.

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

Il DataSource può essere visto come il gestore del trasferimento del contenuto media. La creazione richiede la specifica del protocollo e la locazione dalla quale il media deve essere ottenuto. Mediante queste specifiche, il Manager segue un processo iterativo che consiste nel cercare quel DataSource che supporta il protocollo e il formato dati.
Un punto chiave di tutti i DataSource è che essi non possono essere riusati.
La creazione di un DataSource avviene mediante il metodo ***createDataSource*** che accetta un *URL* o un *MediaLocator* che rappresenta la locazione in cui è presente il media.
Il DataSource può essere classificato su due assi- uno su come il trasferimento dei dati viene inizializzato (push o pull) e l'altro sull'unità di trasferimento (bytes o oggetto Buffer). Utilizzando un *pull* DataSource significa che il trasferimento è inizializzato e controllato dal lato client. Esempi di questo tipo di protocolli include http, ftp, e file. Utilizzando, invece, il *push* DataSource il trasferimento viene inizializzato dal server. Esempi del push media includono il broadcast, il multicast, e il video on demand.

La classe astratta DataSource rappresenta l'astrazione dei protocolli di basso livello che gestiscono il trasferimento dei dati multimediali. DataSource e SourceStream permettono di usufruire di funzionalità particolari non presenti nei normali stream (InputStream), ma utili nella trasmissione e nella gestione di file multimediali come l'accesso random ai dati e la possibilità di suddividere un flusso in "porzioni minori". DataSource, implementando l'interfaccia Controllable, può essere controllata attraverso opportuni controlli ottenibili con i metodi getControl e getControls.
Attraverso i metodi:

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

è possibile gestire il trasferimento dei contenuti multimediali:il primo apre una connessione con una sorgente individuata da locator (parametro passato nel costruttore della classe DataSource(String

locator)), per mezzo della start si può avviare il trasferimento dei dati che può essere interrotto con il metodo stop. Infine, con una chiamata a disconnect viene rilasciata la connessione aperta. Sono inoltre previsti i metodi:
public abstract String getContentType()
public String getLocator()
che ritornano, rispettivamente, il tipo del contenuto multimediale restituito dal DataSource e il locator associato al DataSource stesso.

2.9.3.4 Control

Un oggetto Control è usato per controllare alcune funzioni di esecuzione dei media. I Controls sono ottenuti da Controllable; è l'interfaccia Player che estende Controllable. L'implementazione di un Player può usare l'interfaccia Control per estendere le sue funzioni di processing del media. Per esempio, un Player può esporre un VolumeControl per permetter che il livello del volume venga settato.

Molteplici controlli possono essere implementati con lo stesso oggetto. Per esempio, un oggetto può implementare sia il VolumeControl che il ToneControl. In questo caso, l'oggetto può essere usato per controllare la generazione sia del volume che dei toni.
I controlli sono costituisco le azioni che possono essere svolte su determinati tipi di contenuti multimediali e vengono resi disponibili al Player solo dopo che è stato risconosciuto il "content-type" della presentazione. Per ogni tipo di presentazione vi si possono associare solo determinati controlli.

2.9.4 Java Media Framework

Per lo sviluppo dell'applicazione, in particolar modo per la realizzazione di alcune funzionalità svolte dal lato server, è utile sottolineare l'utilizzo di alcuni concetti fondamentali supportati da queste api.

2.9.4.1 Caratteristiche del JMF

Il Java Media Framework sono API create per permettere di incorporare tipi di dati multimediali in applicazioni o applet Java; si tratta di un pacchetto opzionale, installabile per estendere le funzionalità della

piattaforma JAVA2SE™. Questo componente è stato sviluppato congiuntamente da Sun e IBM, ed è nato con l'intento di fornire un supporto per i più comuni standard di memorizzazione dei dati multimediali, quali: MPEG-1, MPEG-2, Quick Time, AVI, WAV, AU e MIDI.
La peculiarità di Java è l'utilizzo di una Java Virtual Machine (JVM) che interpreta il byte-code generato dal compilatore Java. Questo meccanismo permette la portabilità del codice Java su più piattaforme ma, nel caso in cui sia richiesta un'elevata velocità di esecuzione, impone dei seri vincoli di prestazioni. Il trattamento di dati Multimediali è uno dei casi in cui è richiesta un'elevata velocità computazionale (decompressione delle immagini, rendering, ecc.), e quindi, dove non è sufficiente la sola emulazione della CPU per ottenere delle prestazioni ottimali.
Uno sviluppatore che desidera implementare un lettore multimediale in Java, e vuole ottenere delle prestazioni eccellenti deve ricorrere a codice nativo della piattaforma alla quale è interessato. Questo procedimento comporta due problemi:

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

L'API JMF tenta di risolvere questi problemi, mettendo a disposizione una serie di chiamate ad "alto livello" per la gestione del codice nativo. Usando JMF, l'applicazione non ha necessità di conoscere quando e se deve sfruttare particolari metodi nativi per svolgere una determinata azione. JMF 2.1.1 rende disponibili delle classi che permettono lo sviluppo di applicazioni per la cattura di dati multimediali, fornendo inoltre ai programmatori un controllo addizionale sull'elaborazione e la riproduzione dei dati stessi. JMF è stato progettato per:

1. gestire Audio e Video come oggetti media
2. mettere a disposizione un player JMF per la riproduzione di dati multimediali
3. facilitare la programmazione
4. salvare dei media(su un file)
5. catturare dei media da mezzi come microfoni o telecamere

6. ricevere/trasmettere degli stream media (attraverso la rete)
7. effettuare una transcodifica (cambiare formato) ad un media
8. permettere lo sviluppo di demultiplatori, codificatori, elaboratori, multiplatori e riproduttori personalizzati (JMF *plug-in*).

2.9.4.2 Confronto tra Java Media Framework e MMAPI

La JMF supporta anche addizionali tipi di media e permette di eseguire azioni di rendering e processing.
Le API della JMF 1.1 permettono ad un programmatore di sviluppare programmi Java che eseguono media time-based. Le API JMF 2.0 estendono il framework per provvedere un supporto alla cattura e alla memorizzazione dei dati, per controllare i vari tipi di processing che vengono compiuti durante l'esecuzione. In più, la JMF 2.0 definisce un plug-in che permette di estendere le funzionalità della JMF. La JMF 2.0 supporta la cattura dei media ed indirizza le richieste agli sviluppatori di applicazioni, che vogliono controlli addizionali sulle modalità di svolgimento dei media e sul rendering.
La JMF 2.0 è stata realizzata per:
° essere di facile utilizzo ai programmatori;
° supportare la cattura dei media;
° rendere possibile lo sviluppo dello streming dei media e per conferire le applicazioni in Java;
° rendere possibile una tecnologia capace di implementare delle soluzioni basate su un'estensione delle API e integrare facilmente nuove prospettive con un framework esteso;
° provvedere l'accesso ai dati media;
° rendere possibile lo sviluppo di demultiplexer, di codificatori, multiplexer, e renderer (plug-in JMF);
° mantenere la compatibilità con la JMF 1.0.

La JMF provvede un'architettura unificata e un protocollo per gestire l'acquisizione, la gestione di dati media time-based. E' realizzata per supportare degli standard di tipi come AIFF, AVI, GSM, MIDI, MPEG, QuickTime, RMF, WAV.
La JMF provvede una piattaforma di API Java per accedere ai media framework. L'implementazione JMF può azionare le capacità di un sistema operativo, mentre gli sviluppatori possono creare facilmente programmi Java portabili. Con la jmf è possibile creare anche applet e

applicazioni che presentano, catturano, manipolano e salvano dei media. Il framework rende possibile la gestione dei dati media ed estende la JMF per supportare i tipi e i formati, per ottimizzare la gestione dei formati già utilizzati e creare nuovi meccanismi di presentazione.
Le MMAPI sono state influenzate dalla realizzazione della JMF, e i due tipi di API condividono una serie di somiglianze. Entrambi sono delle API di alto livello, utilizzate per lavorare con media time-based. In particolar modo, condividono i concetti di Player e DataSource (che verranno spiegati in seguito), e prevedono delle funzionalità simili.
A differenza delle MMAPI, la JMF è realizzato per esser utilizzato su un ambiente server o client dove c'è a disposizione una CPU capace di eseguire le applicazioni Java. Il JMF 2.1 ha anche delle funzionalità che non sono presenti nelle MMAPI come la possibilità di codificare e trasmettere (stream) i dati.
Ha anche delle caratteristiche che lo rendono inadatto per essere usato in un dispositivo comune. Per esempio, date le limitazioni del CLDC, i piccoli dispositivi non possono supportare la virgola mobile e un numero di oggetti Java nelle specifiche JMF.
La dimensione del pacchetto api è anche un problema: JMF non può essere ridotto per essere accettato sui 20 KB richiesti da una piattaforma. Per tale motivo è stata necessaria la richiesta di nuove API per tali scopi. Le MMAPI sono utilizzate per risorse client limitate, dove dovranno interagire sicuramente con l'hardware e il software. A differenza del JMF, le MMAPI non effettuano l'ingradimento di immagini e non sono usate per realizzare applicazioni in 2D o 3D. C'è poco beneficio nell'usare le MMAPI in un ambiente che può invece supportare il JMF.

2.9.4.2 Concetti fondamentali del JMF

E' utile sottolineare il ruolo svolto da alcuni componenti che sono di fondamentale importanza per la realizzazione di determinate funzionalità.

Processor

Una delle estensioni desiderabili per un'applicazione sarebbe quella di fornire delle statistiche sulle caratteristiche di trasmissione di un media: velocità di trasmissione, dimensione, durata, tipo di contenuto, codifica.

Comunque, usando un oggetto player, molte di queste informazioni non possono essere rese disponibili perché un player non provvede alcuni di questi controlli sopra indicati. Un'alternativa è quella di usare un oggetto processor che è un tipo di player specializzato e permette di effettuare dei controlli sul processo che esso stesso sta eseguendo.
L'uscita di un processor può essere un DataSource o il rendering del media stesso.

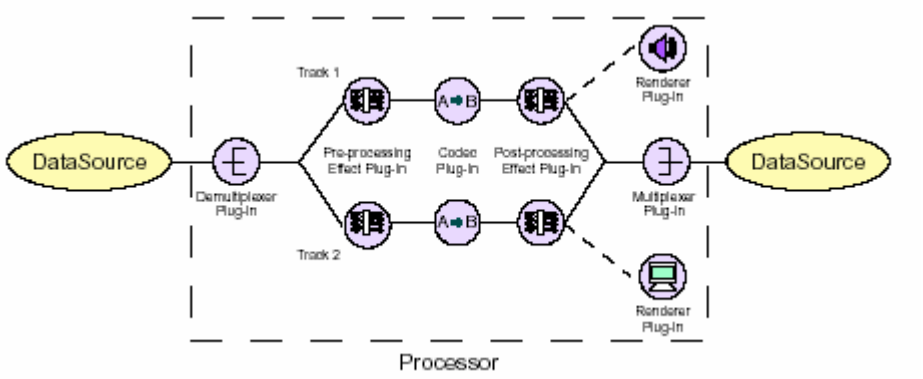

Figura 12: stadi di un processor

Per permettere ad un processor la riproduzione dei dati (played), occorre che il metodo *setContentDescriptor* passato sia *null.*
In questo caso, il materiale multimediale viene salvato all'interno di un file, viene poi spedito attraverso la rete e alla fine del percorso viene direttamente visualizzato sul terminale video.

I processor sono create nello stesso modo in cui vengono creati i Player. Il Manager provvede un addizionale modo per creare il Processor ed è usando il metodo createRealizedProcessor. Viene usato un ProcessorModel che richiede le specifiche di input e output di un Processor. Il metodo createRealizedProcessor prende come argomento un ProcessorModel e crea un Processor che aderisce al ProcessorModel dato. Il Processor appena creato si trova nello stato Realized.
Se il Manager fallisce nella creazione, viene lanciata una NoProcessorException. Se vi sono problemi nella creazione e nella

realizzazione di un Processor, verrà lanciata una IOException o una CannotRealizeException.

Programmare un processor richiede prima la conoscenza del formato del media che sarà processato. La JMF provvede due approcci mediante i quali può essere programmato il processor. Un approccio (che è quello utilizzato in questo progetto) prevede la possibilità di incapsulare tutto ciò che è necessario in un **ProcessorModel**:

```
model=new ProcessorModel(outputFormats,outputContainer);
```

dopo ciò, si dispone della classe Manager mediante la quale è possibile realizzare il processor:

```
processor2=Manager.createRealizedProcessor(model);
```

Mediante la creatRealizedProcessor, che accetta come parametro un processorModel, si crea un processor che si trova già nello stato relized.

E' spesso desiderabile mantenere una copia permanente del media, salvandolo come file. Così il media è disponibile per successive rappresentazioni, processi. Il media originale da dover salvare potrebbe essere stato catturato da un microfono o una telecamera. In JMF, tutte queste istanze sono rappresentate come *DataSource*, e la classe usata per salvare il media come file è il *DataSink*.

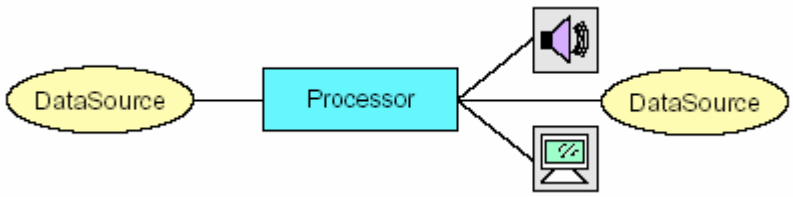

Figura 13: modello di un processor

DataSink

L’interfaccia DataSink specifica un oggetto che accetta i media da una sorgente DataSource e li restituisce ad una destinazione. Molto comunemente al destinazione è un file.
Un oggetto DataSink è creato mediante la classe Manager con il metodo createDataSink(). Tale metodo accetta due parametri: il DataSource al quale il DataSink si connetterà e il MediaLocator che specifica la destinazione.

```
sink=Manager.createDataSink(source,sinkLocation);
System.out.println("creato datasink");
```

Il trasferimento è gestito mediante dei metodi. Con il metodo **open**() si apre una connessione verso la destinazione (specificata dal MediaLocator). Dopo che è stata stabilita la connessione, il trasferimento può iniziare invocando il metodo **start**().

.

Capitolo 3

Analisi del sistema

3.1 Descrizione del progetto

La parte sperimentale della tesi prevede la realizzazione di un sistema orientato alla qualità di servizio per la distribuzione di contenuti video. Grazie a questo sistema sarà possibile realizzare una rappresentazione di contenuti multimediali in maniera tale da venire incontro alle esigenze di un'utenza che richiede di reperire le proprie informazioni mentre si sta movendo fra luoghi diversi. La realizzazione del progetto ha portato ad approfondire alcuni concetti legati alla qualità e alla piattaforma dei dispositivi J2ME con la sua estensione MMAPI.
Prima di passare alla descrizione della fase di sviluppo e alle scelte implementative, è utile per la comprensione del progetto realizzato, introdurre di alcuni concetti fondamentali.

3.2 Scopo del progetto

L'architettura sviluppata fa uso di un modello cliente-servitore.

L'obiettivo che si vuole raggiungere nella trattazione di questa tesi, è quello di poter supportare il movimento di utenti tra diverse località.
Ipotizzando che il cliente si muova dal proprio terminale verso un altro luogo, si vuole assistere il movimento dell'utente; l'idea di fondo da cui parte la realizzazione di questo progetto è quella appena accennata, e per progredire verso tale direzione, si è partiti col realizzare un protocollo di download della risorsa da remoto, con la possibilità di visualizzarla su locale senza la necessità di disporre di tutta la risorsa completa, ma solamente una traccia alla volta e di effettuare lato client il merging delle tracce. Questa scelta è stata adottata, prima di tutto, per venire incontro ai limiti di memoria presenti in un dispositivo portatile in quanto la memorizzazione di tutta la risorsa in locale non sarebbe stata possibile o sarebbe stata consentita solamente per delle risorse dalle piccole dimensioni. La scelta permette di realizzare altri obiettivi, come

quelli legati alla gestione di una qualità di servizio in quanto si vuole garantire che l'utente abbia sempre una corretta e sequenziale visualizzazione della risorsa anche di fronte ad interruzioni della connettività wireless; inoltre, effettuando il reperimento della risorsa con una traccia per volta, si evita la congestione della rete che è possibile, appunto, di fronte a possibili cadute della connessione.
Il risultato è che quello che agli occhi dell'utente appare come uno streaming, alla fine non è altro che un trasferimento di file supportato da semplici protocolli come l'http.
Il lato server, si occupa di gestire le risorse che, come vedremo in seguito, riesce a salvare come file locali e a renderli disponibili al client non appena quest'ultimo avanza le proprie richieste. Tutto ciò può avvenire dopo un semplice controllo iniziale, effettuato per verificare che le risorse richieste siano disponibili e accessibili.
Dopo questa visuale del progetto, si passa a descrivere l'architettura dei realizzazione dell'insieme, per rendere in maniera più evidente l'analisi del sistema.

3.3 Architettura del progetto

La realizzazione del progetto è basata su un'architettura che prevede due entità, cliente e servitore.
L'organizzazione progettuale prevede che le entità entrino in comunicazione tra loro, prima per una verifica iniziale dei dati che essi intendono scambiarsi (fase di inizializzazione o di negoziazione, come mostrato nel punto 1 di fig.14) che viene realizzata mediante una connessione socket, e in seguito per lo scambio effettivo dei dati (fase di trasferimento del materiale multimediale, punto 2 di fig.14) che il client ha richiesto ed ha ottenuto una disponibilità da parte del server, il quale ha accettato di intraprendere l'interazione con il pari, tenendo conto che in quest'ultima fase il protocollo che fa da supporto è l'http.
Occorre mettere in evidenza, come mostra anche al figura, che per ogni presentazione richiesta, vengono aperte n connessioni, quanti sono i pezzi costituenti il media e presenti sul server web. Tali connessioni non vengono aperte contemporaneamente, ma in maniera sequenziale, di modo che all'arrivo di ciascun pezzo sul lato client si abbia il tempo della creazione del player relativo che lo deve gestire.

Adesso non entreremo ancora nel dettaglio descrittivo di ciascuna entità, ma sarà data una visuale di alto livello dell'insieme delle parti e del ruolo che comunque spetta a ciascuna di esse, sottolineando anche le diverse fasi di comunicazione che instaurano e che intraprendono per le loro necessità.

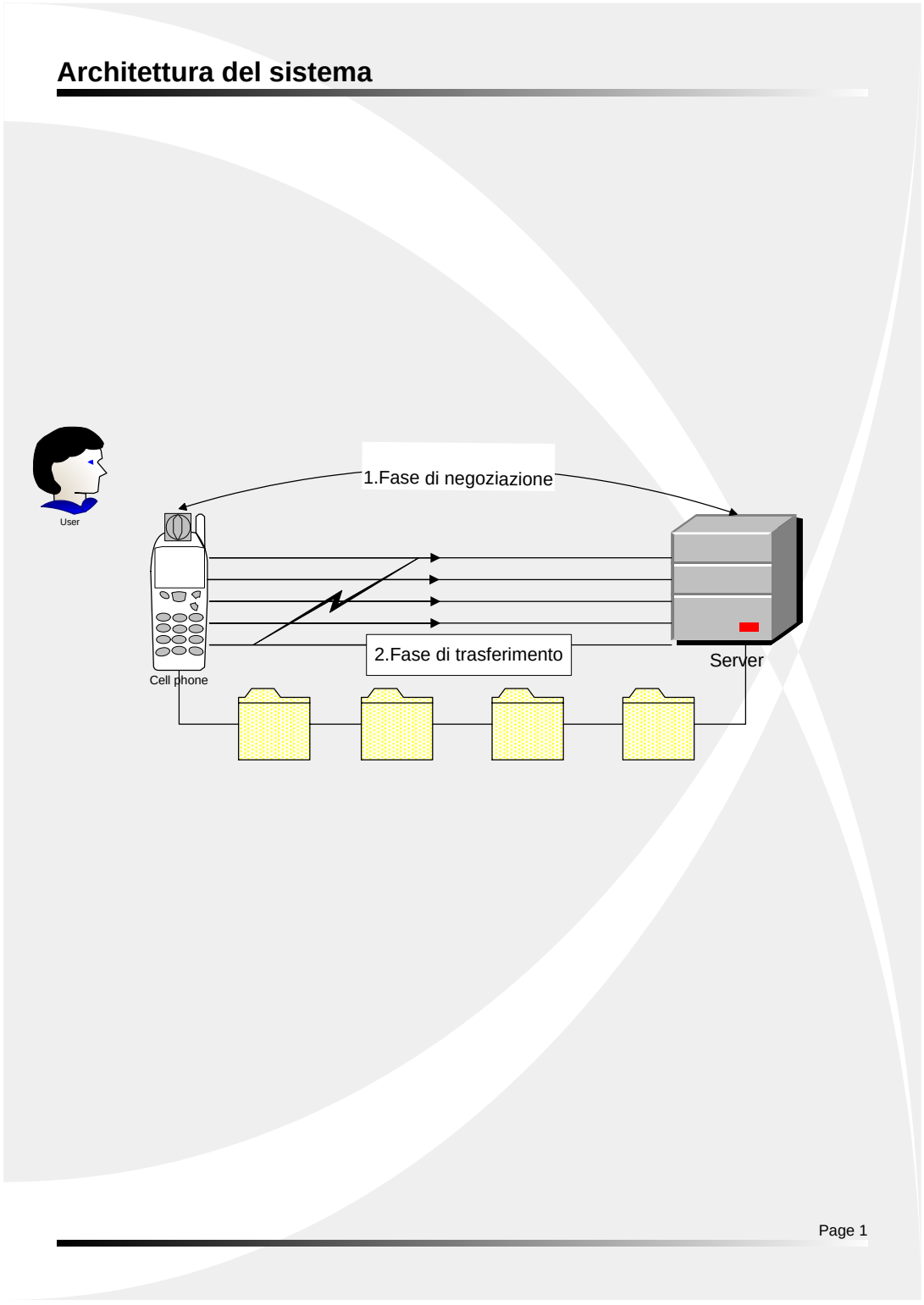

Figura 14: architettura del sistema

3.3.1 Visione delle entità: client-server

Adesso sarà dato spazio alla realizzazione architetturale del progetto, partendo da una visuale che tiene conto del ruolo svolto da ciascuna entità, per scendere in una visuale che le vede operanti nella comunicazione.
Il **server**, una volta che è stato lanciato su una certa macchina, verrà utilizzato per servire tutte le richieste che da quel punto in poi arriveranno a quella macchina per quel tipo di server.
Il server è stato realizzato come Processor, e ciò spiega l'uso della JMF, per venire incontro a delle funzionalità che gli vengono richieste e che sono indispensabili per portare avanti e realizzare la scelta progettuale che è stata avanzata, e cioè quella di poter effettuare il merging lato client solo se sul lato server la presentazione è stata separata in diverse parti.
Dunque, realizzando il server come Processor, inteso come una estensione del server web classico, gli si affida il compito di salvare la presentazione come file, in seguito allo spezzettamento della stessa in diverse parti. Così facendo, si rende disponibile al client la possibilità di effettuare uno streaming della presentazione, che consiste nel trasferimento e nella presentazione, appena esso arriva, di un media da un remoto.
Il fatto di gestire anche questa funzionalità, rende il server più complesso del solito, ma si può usufruire di una efficienza nella gestione delle risorse e nell'intera comunicazione, senza la quale non si potrebbe far uso dei vantaggi che possono derivare dalla scelta applicativa.
Il **client** è l'end-point che riceve i flussi multimediali richiesti. Appare come un'entità fissa sulla macchina dalla quale l'utente effettua l'accesso al sistema. Il fatto che il client non sia un agente non implica poi che il software richiesto per l'esecuzione del client stesso debba già essere presente sulla macchina dalla quale il client verrà lanciato, potrà infatti essere scaricato a runtime.
Per la realizzazione del client si è fatto uso delle MMAPI che dispongono della classe *Player* per visualizzare i media, come è gia stato descritto (vedi paragrafo 2.9.4.1).
Si vuole sottolineare il ruolo che svolge il client nell'ambito architetturale, da cui si potrà evidenziare che è abbastanza ampio. Dal client partono tutte le possibili richieste di una o più risorse che, come

abbiamo visto, se sono disponibili sul lato server, la cui verifica avviene mediante la fase di inizializzazione, possono essere reperite instaurando la connessione http. A questo punto, avviene il trasferimento delle tracce sul lato client che riesce a gestirle in maniera efficiente grazie alle funzionalità offerte dal player, il quale permette, una volta ottenuti i relativi controlli sulla risorsa, di effettuare al visualizzazione. La scelta progettuale prevede che di fronte all'arrivo della prima traccia facente parte della risorsa ne venga fatta la visualizzazione e che nel frattempo si possano accettare le altre mediante l'utilizzo di un altro player che fa da buffer. In questo modo si può realizzare un merging delle varie tracce che arrivano sfruttando l'alternanza dei due player che gestiscono le richieste fino a quando si hanno tracce costituenti la risorsa.
Nella scelta architetturale di ciascuna entità, che sarà descritta nel seguito, verrà approfondito questo concetto e le motivazioni che hanno spinto a tale organizzazione.
E' utile anche evidenziare le fasi di **comunicazione** che vedono coinvolte le due entità, delle quali ci fa comodo distinguere la fase di inizializzazione e la fase di connessione http.
La fase di inizializzazione è realizzata mediante una connessione socket che permette di effettuare un accordo iniziale tra le parti, il che consiste nella verifica della disponibilità della risorsa richiesta sul lato server e di prevedere anche quante tracce costituenti la risorsa dovranno essere scaricate dal client per effettuare un completo merging della risorsa stessa. Se tale fase termina con esito positivo, il che significa che la risorsa che è stata richiesta è disponile e che le tracce rispettive sono presenti e salvate come file, allora è possibile instaurare la connessioni http che permettono l'effettivo reperimento della risorsa da parte del client, con la parallela fase di visualizzazione della stessa; il tutto rende l'architettura molto snella e di limitata complessità.

3.4 Architettura lato server

Adesso si ritiene opportuno entrare nel dettaglio per quanto riguarda la scelta architetturale che vede qui in questione il server.
Il server è stato realizzato mediante l'ausilio della JMF, il che ha permesso di estendere un classico server web con funzionalità che erano indispensabili fornire a tale entità. Il server ha sicuramente il ruolo fondamentale che è quello di gestire delle risorse e su di esse può

effettuare delle operazioni o meno, per renderle più idonee a particolari scopi o per migliorarne l'efficienza o l'utilità; questo è quello che si è voluto fare per poter portare avanti le scelte progettuali e gli obiettivi che caratterizzano proprio questa tesi. Mediante l'ausilio delle JMF, si è potuto estendere sì le funzionalità del server, con l'obiettivo di poter intervenire sulle presentazioni in esso presenti, per poterle renderle disponibili secondo l'esigenza che veniva richiesta dagli obiettivi posti fin dall'inizio. La possibilità è stata quella di poter intervenire sulla presentazione stessa, di manipolarla secondo le nostre esigenze e di gestirla al fine di ottenere dei risultati desiderati.
Potendo intervenire sulla presentazione, si è riusciti a salvare la stessa come file con la precedente separazione in diverse parti; il che ha permesso di rendere disponibile la presentazione lato client mediante successivi trasferimenti, evitando indesiderati inconvenienti legati alla visualizzazione del media di fronte a possibile cadute della connessione wireless. Occorre tenere presente che prima della fase dell'effettivo trasferimento del media si ha una fase di accordo iniziale che viene realizzata lato server mediante una connessione socket. Grazie a questa connessione, in seguito alla quale il server rimane in attesa di richieste, si può gestire un accordo tra le due parti relativo alla disponibilità del server della presentazione che viene richiesta dal client, e prevede che in caso di piena disponibilità, il server effettui una ricerca della stessa rendendo noto il numero di pezzi che dispone della presentazione in questione.
Così facendo, si migliora l'efficienza del server, rendendo anche più snella l'attività lato client, che può organizzare le proprie risorse per un numero già dichiarato di pezzi che dovranno essere gestiti.
L'architettura del server è di facile realizzazione, la quale prevede, come già detto, che vengano gestite due funzionalità:
° accordo tra le parti mediante connessione socket (come evidenziato nel punto 1 della fig.16);
° invio materiale multimediale mediante connessione http(come nel punto 2 della fig.16); è da evidenziare, come già sottolineato nella descrizione dell'architettura generale, che la connessione http viene esplicata tante volte quanti sono i pezzi che costituiscono la presentazione che è presente sul web server stesso.
° spezzettamento della presentazione e memorizzazione delle stesse come file (come evidenziato nel punto 4 di fig.16).

La realizzazione di queste permette di avere un server efficiente al punto giusto per gli obiettivi posti, avendo esteso delle funzionalità che comunque non rendono pesante l'applicazione.

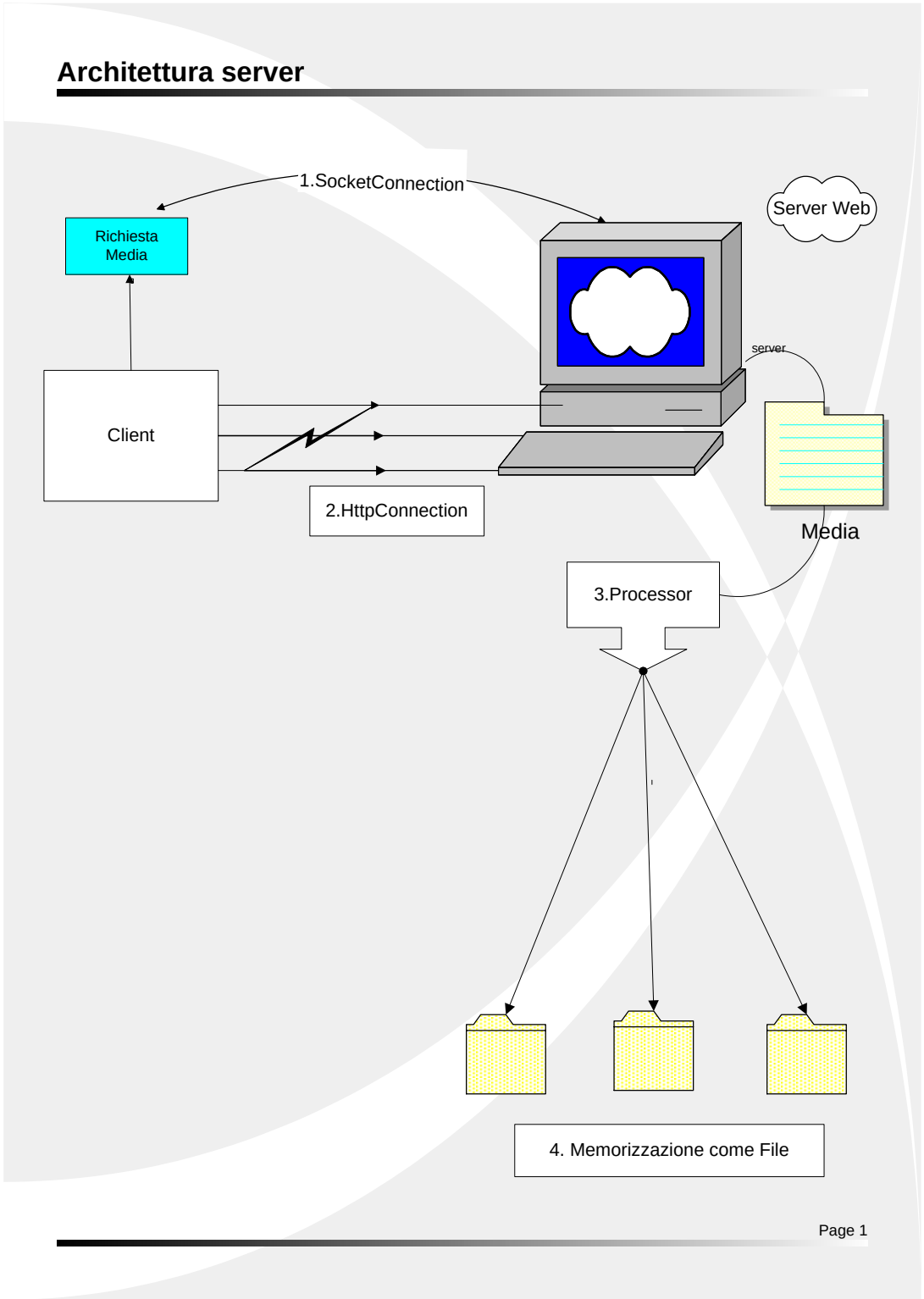

Figura 16: architettura server

Come si evince dalla figura, il server presenta un'estensione che gli permette di effettuare la memorizzazione della presentazione come file e con il precedente spezzettamento della stessa in varie parti; questo è possibile con l'ausilio della JMF che permette di utilizzare l'oggetto processor che dispone di metodi adatti alla realizzazione di tali funzionalità. Tutto questo verrà descritto nella fase implementativa e sarà di più facile comprensione.

3.5 Architettura lato client

La realizzazione progettuale vede il ruolo svolto dal lato client, i cui compiti sono essenziali ma articolati.
Il client è l'entità che riceve i flussi multimediali, li gestisce, li coordina e ne effettua la visualizzazione su display. Tutto ciò è realizzabile grazie all'ausilio del *Player* che permette la gestione delle varie tracce e la visualizzazione delle stesse su display.
Analizziamo con dettaglio l'architettura che prevede la realizzazione dell'entità client che avanza al server le proprie richieste.
La classe dalla quale parte tutta l'applicazione è il *VideoTest*, dalla quale vengono effettuate le richieste della risorsa. Fatto ciò, il progetto prevede un'architettura produttore-consumatore mediante la quale si riesce a gestire il flusso multimediale e l'effettiva visualizzazione del media su display. La classe che gestisce la creazione del player, possibile grazie alla ricezione dell'inputStream relativo alla presentazione, con la conseguente visualizzazione del media mediante merging delle varie parti, è il *VideoPlayer*. Prima della realizzazione del player, occorre realizzare altri passi necessari alla realizzazione del progetto stesso.
Una volta avanzata la richiesta, la classe *Produttore* gestisce prima l'accordo iniziale, per ricevere il numero di pezzi costituenti la presentazione, poi permette l'apertura della connessione http lanciando il thread relativo che è il PlayerReader. A questo punto, dalla classe *PlayerReader* viene gestita la lettura dallo stream della risorsa e, per ogni pezzo ricevuto, si effettua una notifica al Produttore, il quale può rendere disponibile al VideoPlayer l'inputstream necessario alla creazione del player stesso.
Ricevute le varie parti della risorsa, occorre un consumatore che possa reperirle per effettuarne la visualizzazione: in tal caso, il ruolo di

consumatore spetta al player, gestito nella classe VideoPlayer, che ricevuta la presentazione, riesce a gestirla e ad effettuarne la visualizzazione a video. Nel progetto, l'architettura prevede la creazione di due player che riescono a gestire le tracce fino a quando ne arrivano e si prevede che vi sia un'alternanza tra i due nelle fasi di ricezione e visualizzazione delle trame stesse. Questa scelta, cioè quella relativa alla creazione di due soli player, è dovuta al fatto che così facendo si ha un più rapido susseguirsi da una trama all'altra di modo che le varie parti vengano incollate il più possibile; la creazione di più player, ad esempio uno per ogni traccia, avrebbe reso meno efficiente la realizzazione e molto meno flessibile, con il rischio di non ottenere un risultato ottimale.
In presenza di più di una richiesta, è necessaria la creazione di entrambi i player e, all'arrivo della prima traccia viene lanciato subito in esecuzione il primo player mentre il secondo raccoglie la seconda richiesta e aspetta che il primo abbia terminato la propria esecuzione prima di entrare in funzione; stessa cosa dovrà effettuare il primo, al termine della propria esecuzione, se vi sono da gestire altre richieste.
L'applicazione prevede che nonostante vi siano n pezzi, la gestione venga delegata a due soli player che, come appena descritto, riescono ad alternarsi nella fase di visualizzazione delle proprie parti.
Per garantire un buon funzionamento e una visualizzazione che preveda un merging delle varie tracce, si è pensato che ciascun player gestisca una propria coda di richieste, in modo da facilitare la previsione di eventuali richieste e la relativa creazione del player che le deve gestire.
La realizzazione lato client è stata caratterizzata dall'utilizzo di thread che hanno permesso, mediante una sensata sincronizzazione degli stessi, di realizzare la fase di creazione dei player con la preventiva fase di lettura dello stream, delegata al rispettivo thread. Nella realizzazione di questo progetto si può ricordare che lo stato viene mantenuto sul latoclient.
Di fronte a questi risultati si può evincere che l'idea architetturale del client e la gestione delle varie risorse è stata proficua ed ha consentito ad una buona riuscita della realizzazione finale.

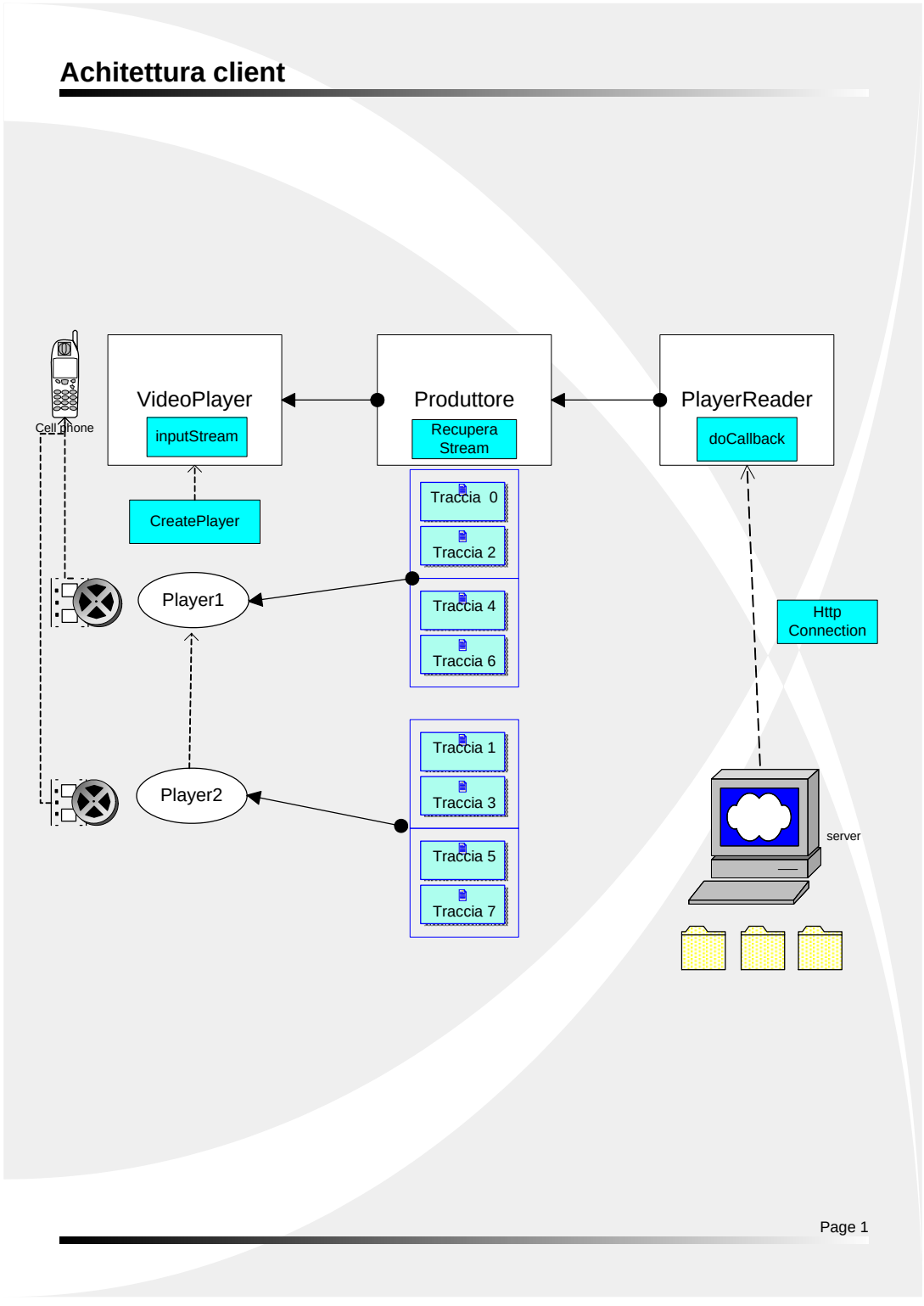

Figura 17: architettura client

3.6 Architettura delle fasi di comunicazione

Dopo aver dato una descrizione dell'idea architetturale delle due entità coinvolte nella realizzazione del progetto, è utile sottolineare come le due parti comunicano tra loro nelle due fasi che, come già è stato accennato, le caratterizzano e cioè la connessione socket e la connessione http.
E' di fondamentale importanza mettere in evidenza i protocolli che sono stati utilizzati nella realizzazione del progetto, sottolineando il perché delle varie scelte, le caratteristiche, i pro e i contro dell'http connection e della socket connection.

3.6.1 Fase di inizializzazione e protocollo SocketConnection

La fase di inizializzazione del sistema è necessaria per stabilire un **accordo** iniziale tra le parti che vogliono intraprendere una comunicazione http per lo scambio di dati. Il servizio di inizializzazione impone un modello per il collegamento fra le varie entità, e impone che in tali entità siano implementati alcuni metodi, lasciando al progettista dell'applicativo il compito di concentrarsi sulla realizzazione delle entità stesse.
Questa fase di inizializzazione è portata avanti facendo uso di una *comunicazione socket*; il client inoltra al server la richiesta di disponibilità di una risorsa, chiedendo, ad esempio, se una risorsa è disponibile e se tale quanti componenti dovrà attendere e dunque scaricare.
Se tale operazione torna un esito positivo, ha senso intraprendere una connessione http che prevede lo scaricamento della risorsa da parte del client, altrimenti il client può effettuare una nuova richiesta al remoto per una nuova risorsa.
Il servizio di inizializzazione provvede anche ad effettuare la fase di negoziazione iniziale della QoS e prenotazione delle risorse. Il remoto, a questo punto, per prima cosa vede se ci sono risorse sufficienti per la presentazione richiesta. Se ci sono sufficienti risorse per la presentazione richiesta, il processo di inizializzazione termina, altrimenti il remoto può vedere se esistono presentazioni alternative a qualità più bassa e tenta di rendere disponibile tale disponibilità.

3.6.2 Fase trasferimento materiale multimediale

La fase di comunicazione vera e propria è quella relativa allo scambio dei dati ed è realizzata mediante una connessione http.
Alla richiesta lato client, segue una verifica da parte del remoto circa la disponibilità della richiesta e in caso di esito positivo si procede al reperimento delle tracce da parte del client. Il protocollo che fa da supporto è il protocollo http, ma si vuole evidenziare come viene gestita la fase di reperimento delle risorse.
Per poter realizzare questo servizio, è necessaria un'interazione tra cliente e servitore. L'interazione tra il cliente e il servitore di questo servizio sarà basato sul modello consumatore-produttore, con il supporto delle notifiche di eventi. Il cliente richiede cioè il downloading di un certo media al servitore; si suppone che il remoto abbia già la risorsa disponibile sotto forma di file e se ciò avviene, il client può proseguire con il reperimento della stessa. La realizzazione del modello prevede che sul lato client vi sia un thread che si occupi della lettura da stream della risorsa e per ogni traccia letta ne faccia una notifica all'eventuale consumatore che può procedere alla visualizzazione della stessa.
Questo modello merita un'attenzione particolare perché, come già evidenziato nell'architettura lato client, è il cuore del progetto che permette la realizzazione della fase progettuale con una certa dimestichezza, avendo impostato quella architetturale in maniera abbastanza solida.

Conclusione

In questo capitolo si è entrati nel vivo della situazione, cercando di mettere in evidenza le scelte architetturali di ciascuna entità e cioè del client e del server, avendo anche sottolineato le fasi che coinvolgono entrambi nelle loro comunicazione. La realizzazione di questo progetto prevede che alla richiesta della presentazione, le varie parti siano già rese disponibili dal server web; una possibile estensione potrebbe essere quella di spezzare online la presentazione, sempre lato server, senza averla tutta disponibile sullo stesso server. Da questa esposizione si passerà nel prossimo capitolo a scendere più nel dettaglio, per indicare con maggiore chiarezza i passi implementativi relativi al progetto e a

dare anche una spiegazione delle scelte che sono state effettuate per garantire dei vantaggi che non sarebbero stati possibili affrontando altre scelte.

Capitolo 4

Implementazione del progetto

4.1 Scelte implementative

In questo capitolo si vuole descrivere un'applicazione che realizza lo streaming di un flusso multimediale. Le entità che verranno descritte nella loro parte implementativa sono il client e il server.
Prima di iniziare la descrizione della realizzazione di ciascuna entità, è utile sottolineare la scelta che è stata affrontata nella modalità di trasmissione del flusso di dati richiesti. Il progetto è stato realizzato con l'obiettivo di poter estendere la funzionalità fino ad n pezzi, gestiti da due soli player. Procediamo con l'ipotizzare che la fase di inizializzazione sia stata portata a termine con successo; segue che la fase di trasferimento della risorsa avviene, come già è stato menzionato nel capitolo precedente, mediante notifiche al client effettuate quando la risorsa è disponibile e dunque pronta per essere rappresentata. Questo processo non prevede che tutto il media richiesto sia disponibile e dunque salvato sul lato client prima di essere rappresentato (played).
Si procede con un rendering delle varie parti che costituiscono il media e che provengono da un web server, di modo che quando la prima parte è già stata notificata al client, questo procede ad effettuare il playing di essa mentre il remoto rende disponibile i successivi pezzi. E'interessante sottolineare l'efficienza che può derivare da tale scelta, considerando che la presentazione multimediale richiesta deve comunque essere accettata da dispositivi come cellulari che hanno delle capacità di memoria molto limitate, permettendo dunque delle opportunità che in caso contrario non sarebbero state possibili.
Con tale scelta implementativa si viene a sottolineare anche il motivo fondamentale per il quale viene affrontata e cioè di poter venire incontro alle richieste di mobilità di utenza che vogliono reperire le informazioni su un device wireless.

4.2 Implementazione della fase di inizializzazione

La fase di inizializzazione prevista tra client e server prevede l'uso di una socket connection per stabilire un accordo iniziale.
Per realizzare tale scelta vengono create delle classi che hanno lo scopo di realizzare tali funzionalità. Per il lato client, la realizzazione di una socket connection è lasciata alla classe ***Produttore***:

```
socketConnection =
        (SocketConnection) Connector.open(connectString);
        Utils.debugOut("Realizzo connessione socket");
```

una volta aperta la connessione, il client effettua la richiesta della presentazione multimediale di cui vuole effettuare il rendering, scrivendo sulla socket connection che è stata aperta:

```
outSock.writeUTF(invio);
    System.out.println("Inviato  " + invio);
```

successivamente, attende la risposta da remoto, leggendo da socket:

```
esito = inSock.readInt();
    System.out.println("Esito trasmissione: " + esito);
```

Questa parte ha sicuramente il corrispondente sul remoto che è stato implementato dalla classe ***Location.*** Questa classe implementa le funzionalità del server e come si vedrà successivamente, nella fase implementativa della connessione http, questa classe prevede anche altre funzionalità.
Per adesso, sottolineiamo il fatto che, il server effettua la creazione della connessione socket specificando il numero di porta sulla quale ascolta:

```
int port = -1;

try {
  if (args.length == 1) {
    port = Integer.parseInt(args[0]);
  } else if (args.length == 0) {
    port = PORT;
```

```
    } else {
System.out .println("Usage: java Server or java Server port");
System.exit(1);
```

Successivamente, si mette in attesa di richieste:

```
serverSocket = new ServerSocket(port);      serverSocket.setReuseAddress(true);
     System.out.println("Server: avviato ");
     System.out.println("Creata la server socket: " + serverSocket);
   }
   catch (Exception e) {
     System.err.println("Problemi nella creazione della server socket:"          +
e.getMessage());
     e.printStackTrace();
     System.exit(2);
   }
   try {
     while (true) {
       Socket clientSocket = null;
       DataInputStream inSock = null;
       DataOutputStream outSock = null;
       System.out.println("\nIn attesa di richieste...");
```

Se questa fase termina con esito positivo, la comunicazione per lo scambio dati può essere intrapresa. Il remoto comincia a rendere disponibile i propri dati non appena la successiva connessione viene aperta dal client e quest'ultimo comincia a reperire le informazioni richieste.

4.3 Implementazione della comunicazione client-server

Il servizio http è stato progettato per essere utilizzato su dispositivi come telefoni cellulari o computer palmari, perciò l'architettura software è strutturata secondo quanto imposto dalla tecnologia J2ME .
Il *servizio http* consente all'utente di accedere ai dati multimediali cui è interessato, in particolar modo consente di accedere ad un file multimediale. La connessione http viene aperta nel momento in cui la fase precedente di connessione socket, utilizzata per un opportuno controllo dei dati, sia avvenuta con un certo successo il che significa che

la presentazione richiesta è disponibile sul server e se ne può prendere atto nel momento in cui il client decide di iniziare il reperimento di essa.
Occorre specificare che Http Service è stato sviluppato per approfondire la struttura e i package della piattaforma J2ME e della sua estensione MMAPI.
In questi paragrafi verranno affrontate le problematiche e le scelte progettuali per lo sviluppo di ogni singolo servizio.

4.3.1 Lato Client

Il lato client ha il compito di recuperare il flusso multimediale e di visualizzarlo sul display del dispositivo mobile, facendo uso dell'opportuna fase di rendering che prevede che la visualizzazione dei vari pezzi del media siano "ricomposte" sul client mediante una corretta e sequenziale visualizzazione di ciascuna di esse. Come già descritto nella fase di analisi, il client è realizzato mediante un Player che viene creato dalla classe Manager.
Nello sviluppo del progetto, la classe che si occupa di questo è il *VideoPlayer:* questa classe sfrutta le funzionalità del package MMAPI.
Si procede adesso alla descrizione del processo implementativo del progetto, ripercorrendo i vari passi che ne prevedono il corretto funzionamento.
L'esecuzione parte dalla classe *VideoTest*, dalla quale vengono passati gli Url relativi alle presentazioni multimediali cui si vuole accedere:

```
for(int i=0;i<requestedUrls.size();i++)
        {
        try
{
        videoPlayer.open((String) requestedUrls.elementAt(i),i);
}
        catch (Exception ex)
{
        ex.printStackTrace();
```

A questo punto, viene invocato il metodo open() della classe VideoPlayer, a cui vengono passati come parametri l'URL della presentazione ed un intero identificativo.

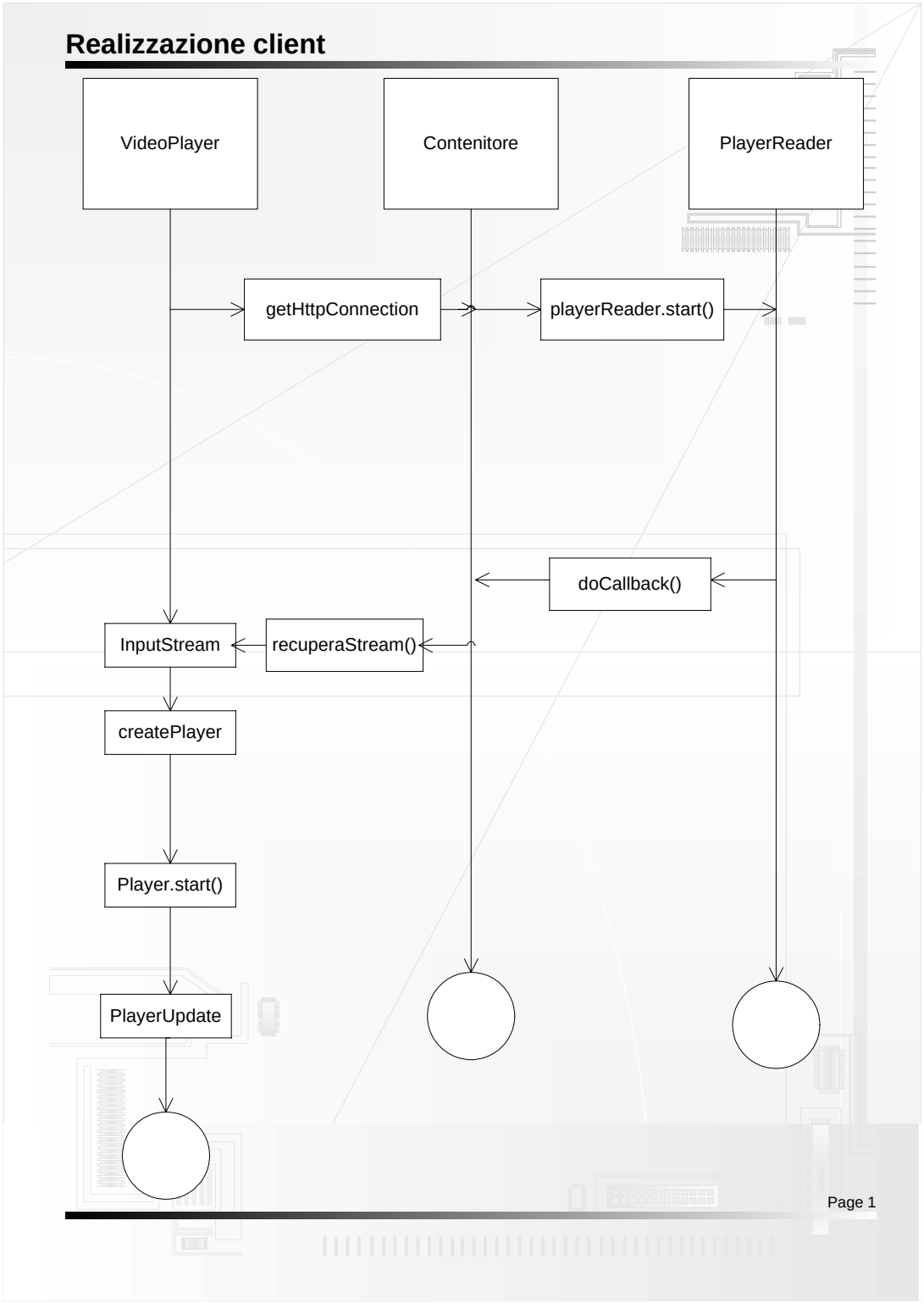

Figura 18: implementazione client

E’ importante sottolineare la scelta adottata per poter gestire n pezzi supportate da due player di cui il primo viene lanciato all’arrivo della prima richiesta mentre il secondo attende il termine del primo per entrare in esecuzione. L’idea è stata quella di realizzare due code, una per ogni player, in base al numero di pezzi che arrivano, dei quali si può tenere traccia mediante l’intero identificativo che viene passato come parametro. Se il pezzo è solo uno, viene creato un solo player, il primo, che lo gestisce; se l’id prevede anche un secondo pezzo, viene creato anche il secondo player. A fronte di un numero di pezzi ulteriore, quelli che hanno un identificatore pari vengono memorizzati in un vettore, in modo da poter essere gestite dal primo player; quelli che hanno un identificativo dispari vengono memorizzati in un secondo vettore, tali da essere gestite dall’altro player.
Dopo la fase di inizializzazione che prevede la conferma di dati richiesti al web server, si passa alla creazione della connessione http mediante l’invocazione del metodo

```
produttore.getViaHttpConnection(url);
```

Quando viene invocato questo metodo, dalla classe Produttore viene lanciato il thread PlayerReader che si occupa di gestire la lettura della presentazione dallo stream:

```
PlayerReader pr=new PlayerReader(this);
pr.start();
```

Una volta lanciato il thread *PlayerReader* , la connessione viene aperta sul web server specificato in cui è presenta la presentazione multimediale e comincia la lettura dallo stream. La logica del progetto prevede che quando un pezzo del media è stata recuperato dallo stream, il thread invoca una callback sul produttore in modo che il player possa ricevere l’inputStream necessario per la creazione del player stesso. A questo punto, il player continua ad effettuare le sue fasi previste e cioè la fase realized nella quale ha disponibili tutti i controlli relativi al media stesso, il successivo prefetch e infine la start.
Vediamo come sia necessario l’inputStream per la creazione del player, che viene recuperato solo quando è disponibile a seguito della notifica:

```
InputStream ins = produttore.recuperaStream();
```

```
try {
player = Manager.createPlayer(ins, "video/mpeg");
} catch (IOException e) {
e.printStackTrace();
} catch (MediaException e) {
e.printStackTrace();
}
```

Dopo la creazione, il player è caratterizzato da un ciclo di vita che prevede delle fasi attraverso cui il player è chiamato ad effettuare (vedi paragrafo 2.9.4.1.1).
Il player è caratterizzato dalla fase realized, e dal prefetched, possibile solo se sono stati effettuati tutti i controlli relativi al player:

```
if ((vidc = (VideoControl) player[p].getControl("VideoControl")) != null) {
videoItem[p] = (Item) vidc.initDisplayModeVideoControl.USE_GUI_PRIMITIVE,
null);
}
else if (logo != null) {
append(new ImageItem("", logo, ImageItem.LAYOUT_CENTER, ""));
}
Control[] controls = player[p].getControls();

for (int i = 0; i < controls.length; i++) {
if (controls[i] instanceof GUIControl && controls[i] != vidc) {
}
if (controls[i] instanceof VolumeControl) {
vc = (VolumeControl) controls[i];
}
if (controls[i] instanceof RateControl) {
rc = (RateControl) controls[i];
}
}
status = new StringItem("Status: ", "");
if (vc != null) {
audioStatus = new StringItem("", "Volume:");
}
player[p].prefetch();
```

Una volta raggiunto lo stato started, il player mette in esecuzione il media che ha ricevuto. E' utile sottolineare come gli ***eventi*** del player salvano informazioni sui cambiamenti di stato del player stesso e sui

controlli del player. Per ricevere degli eventi, un oggetto deve implementare l'interfaccia *PlayerListener* e usare il metodo *addPlayerListener* per registrare il suo interesse a ricevere gli eventi del player.
Quando il player raggiunge un END-OF-MEDIA, viene utilizzato il metodo **playerUpdate** per catturare gli eventi.

```
public void playerUpdate(Player plyr, String evt, Object evtData) {
if (evt == END_OF_MEDIA) {
```

Questo metodo viene chiamato per inviare un evento al listener quando si verifica un relativo evento del player.
Nel momento in cui il primo player raggiunge il termine della propria esecuzione, si verifica un end_of_file e il player viene chiuso; l'evento end_of_file viene catturato e dal metodo playerUpdate viene lanciata l'esecuzione del player cui spetta l'esecuzione:

```
// Fa partire il prossimo
if (nextPlayer != null)
{
nextPlayer.start();
Utils.debugOut("[PlayerUpdate] lancio la start" );
}
```

E' utile sottolineare che per effettuare la visualizzazione del media sempre su un unico frame, occorre eliminare il frame relativo al player che ha appena terminato l'esecuzione e il relativo append del frame per il player corrente è stato creato nella fase di inizializzazione del player stesso.

```
vd = videoItem[playerIndex].getLayout();
delete(vd);
```

Si deve tener presente che se sono presenti più di due pezzi costituenti il media, occorre interrogare il vettore relativo al player che ha terminato per primo, per vedere se ha ancora dei pezzi in coda da gestire:

```
if(currentPlayerIndex % 2==0){
if(requestVector1.size()!=0){
nextPlayer=CreatePlayer((String)requestVector1.elementAt(0),nextPlayer);
```

```
requestVector1.removeElementAt(0);
}
```

Se tale player ha ancora dei pezzi in coda da gestire, viene effettuata la creazione, tenendo presente che al termine della sua ultima esecuzione era stato chiuso; per la creazione viene passato come parametro la stringa relativa all'url da cui verrà ricavata la presentazione. Tutto ciò avviene mentre è ancora in esecuzione il secondo player e quando quest'ultimo terminerà, verrà prima deallocato e verrà effettuato lo stesso procedimento appena descritto anche per esso.
Il tutto si basa su una alternanza dei due player che gestiscono le rispettive code fino a quando vi sono dei pezzi in attesa di essere rappresentati.

Si vuole sottolinerare come tale processo prevede la sincronizzazione di *threads.*
Nel momento in cui, dal VideoPlayer, viene invocato il metodo produttore.getViaHttpConnection(), viene lanciato il thread playerReader

```
PlayerReader pr=new PlayerReader(this);
pr.start();
```

che si occuperà di recuperare i dati dallo stream e contemporaneamente continua l'esecuzione del thread relativo al VideoPlayer.

Il PlayerReader apre la connessione http, legge dallo stream e per ogni pezzo ricevuto effettua la notifica al client:

```
try {
   {
HttpConnection hc = (HttpConnection)Connector.open(url[indicePlayer]);
InputStream is = hc.openInputStream();
      long len = hc.getLength();
            baos.reset();

while ((len4 = is.read(buf)) >0) {
baos.write(buf, 0, len4);
}
Utils.debugOut("Faccio la callback");

produttore.doCallback(baos.toByteArray());
```

Quando viene invocata la doCallback del produttore, i dati sono disponibili e solo a questo punto la
InputStream ins = produttore.recuperaStream() viene sbloccata dall'attesa.
Senza un'opportuna sincronizzazione dei thread, il rischio sarebbe quello di invocare la richiesta si stream senza che questo sia ancora disponibile o di perderlo perché il primo thread in esecuzione è più veloce dell'altro.

4.3.2 Lato Server

Il server è l'altra entità fondamentale per la realizzazione di questo progetto. La classe che gestisce le funzionalità svolte dal server è la *Location2*.
Il server inizia con il gestire la fase d'accordo iniziale mediante la creazione di una socket stream:

```
Socket clientSocket = null;
      DataInputStream inSock = null;
      DataOutputStream outSock = null;

      System.out.println("\nIn attesa di richieste...");
      try {
        // bloccante finchè non avviene una connessione
        clientSocket = serverSocket.accept();
```

Una volta creata la socket, il server rimane in *attesa* di *richieste* da parte del client. Questa fase serve infatti al client per interrogare il server circa le presentazioni messe a disposizione da quest'ultimo e la sua disponibilità a distribuirle. Se questa fase iniziale ha esito positivo, ha senso intraprender la connessione http mediante la quale il client recupera la presentazione richiesta.
Iniziamo con l'analizzare prima la realizzazione del server e in seguito le funzionalità svolte da esso.
Il server è creato mediante il Processor. Programmare un processor richiede prima la conoscenza del formato del media che sarà processato. La JMF provvede due approcci mediante i quali può essere programmato il processor. Un approccio (che è quello utilizzato in questo progetto) prevede la possibilità di incapsulare tutto ciò che è necessario in un **ProcessorModel**:

```
model=new ProcessorModel(outputFormats,outputContainer);
```

dopo ciò, si dispone della classe Manager mediante la quale è possibile realizzare il processor:

```
processor2=Manager.createRealizedProcessor(model);
```

Mediante la creatRealizedProcessor, che accetta come parametro un processorModel, si crea un processor che si trova già nello stato relized.

E' spesso desiderabile mantenere una copia permanente del media, salvandolo come file. Così il media è disponibile per successive rappresentazioni, processi. Il media originale da dover salvare potrebbe essere stato catturato da un microfono o una telecamera. In JMF, tutte queste istanze sono rappresentate come *DataSource* (vedi paragrafo 2.9.5.2), e la classe usata per salvare il media come file è il *DataSink*.

4.3.2.1 Location2

La classe Location2 è la responsabile della gestione dei media e dell'eventuale salvataggio di essi come file. Il costruttore di questa classe specifica la sorgente e la destinazione dei media (entrambi medialocator o stringhe), il formato desiderato per ciascun pezzo del media e il descrittore per il contenitore del media risultante.

```
outputFormat[0]=new VideoFormat(VideoFormat.MPEG,new
Dimension(160,120),0,Format.byteArray,1.0f);
container=new FileTypeDescriptor(FileTypeDescriptor.MSVIDEO);
loc=new Location2(ss,ds,outputFormat,container,1.0f);
```

Come specificato sopra, è possibile notare che il formato di output desiderato è un video format, in particolare un formato MPEG, con la specifica di essere salvato come byte array.

Questa classe illustra il significato di trattare le caratteristiche del JMF. La classe ascolta gli eventi sia del processor che del dataSink e mantiene il loro modo interno o il loro stato corrente. Questo è esposto mediante una serie di costanti e mediante il metodo getState(), che è usato per verificare lo stato corrente.

Per evidenziare la possibilità di salvare una presentazione come file locale, è stato necessario introdurre il concetto di dataSink; dopo la creazione e la realizzazione di quest'ultimo che prevede come parametri un mediaLocator, indicante la destinazione della presentazione salvata come file, invocando i metodi open e successivamente la start si inizia il trasferimento della presentazione nella nuova destinazione.
Il trasferimento verso la nuova destinazione avviene invocando il metodo transfer(), in cui il procedimento descritto viene svolto.

```
sink = Manager.createDataSink(processor2.getDataOutput(),sinkLocation);
sink.open();
sink.start();
```

Dopo aver evidenziato la possibilità di salvare la presentazione come file, si vuole indicare una possibile funzionalità permessa al server grazie all'utilizzo delle funzioni messe a disposizione dal processor. Infatti, l'idea sarebbe quella di poter realizzare sul server la divisione in vari pezzi della presentazione disponibile ed effettuare per ciascuna di esse un formato tipo file.
In questo metodo, è possibile creare un oggetto Time e recuperare la durata del media.

```
time=new Time(mediaTimeVecchio);
   mediaTimeVecchio=time.getSeconds();
```

Una volta fatto ciò, si setta il processor sulla base di questo oggetto time e si può effettuare lo spezzettamento del media in diverseparti, attendendo per un tempo di wait e bloccando il processor:

```
processor2.setMediaTime(time);

while(state!=FAILED && state!=FINISHED && waited<timeOut){
try{
Thread.sleep(WAIT_PERIOD);
processor2.stop();
```

Una volta ottenute le varie parti del media, il server si occupa di salvarle come file e di renderle disponibili ad un'eventuale richiesta client.

Si è voluta evidenziare una possibile situazione, una scelta implementativa che comunque non permette manualmente di spezzare il media con questo metodo, ma solo di ridurre la sua durata, cioè di estrapolare un pezzo più piccolo da tutta la risorsa. Con questo metodo, da un media non è possibile ottenere un primo pezzo avuto invocando lo stop sul processor perché, quando si tenta di ottenere pezzi successivi che rappresentano il seguito del media, il file non è più completo o addirittura non si può più ritenere tale in quanto mancano delle informazioni che possono essere anche di header. Dunque, tentare di invocare un altro stop sul processor dal punto in cui è stata troncata la prima parte, per cercare di ottenere la successiva, non permette di ottenere un risultato valido.
Una volta sottolineato ciò, si vuole invece ricordare la funzione del server che rimane quella di gestire comunque i propri dati e di renderli disponibili quando richiesti.

4.4 Gestione del servizio http

Si passa ora ad evidenziare la gestione di una connessione sicura nel momento in cui si possono verificare delle eccezioni o si realizzi una caduta di connessione. Se ciò avviene, si deve garantire che la comunicazioni continui e riprenda dal punto in cui si è fermata.
Per garantire dunque una certa qualità di servizio, si è giunti ad una scelta implementativa che prevede un certo numero di tentativi da effettuare se la connessione cade e occorre ristabilirla. Infatti, se ciò dovesse accadere, si tenta di aprire nuovamente al connessione fino a quando si hanno dei tentativi a disposizione, decrementando di volta in volta il numero di quelli già effettuati. Questo meccanismo viene effettuato per ogni pezzo disponibile e in caso di problemi, i tentativi sono relativi ad ogni pezzo in questione, cioè su quello per il quale si è verificata un'eccezione.

```
while(tentativi>0 && indicePlayer<MaxPlayers)
{
try {

HttpConnection hc = (HttpConnection)Connector.open(url[indicePlayer]);
InputStream is = hc.openInputStream();
long len = hc.getLength();
```

```
        baos.reset();

while ((len4 = is.read(buf)) >0) {
baos.write(buf, 0, len4);
}
Utils.debugOut("Faccio la callback");
produttore.doCallback(baos.toByteArray());
if(hc!=null)
hc.close();

indicePlayer++;
} catch (Exception e) {
Utils.debugOut("vediamo cosa succede" + e);
e.printStackTrace();

tentativi--;
}
```

Si vuole indicare che l'indice del player viene incrementato solo se tutto è andato a buon fine, altrimenti l'eccezione viene lanciata prima e non si arriva ad incrementare l'indice del player successivo in quanto il tentativo deve essere ripetuto per lo stesso player; entrando nel catch, si decrementa il numero dei tentativi a disposizione.

4.5 Processi

E' già stato indicato che per la realizzazione del progetto si è fatto uso dei threads e adesso ci si vuole soffermare un po' sul loro utilizzo e sull'importanza di poter gestire bene la sincronizzazione di essi perché è su questo concetto che si basa la buona riuscita di un buon progetto.

E' possibile che due processi possono **interagire** per:
cooperare: i processi interagiscono allo scopo di perseguire un obiettivo comune;
competere: i processi possono essere logicamente indipendenti ma necessitano della stessa risorsa(file, variabile, dispositivo) per la quale ci sono dei vincoli di accesso.
In entrambi i casi è necessario disporre di strumenti di sincronizzazione.
La **sincronizzazione** permette di imporre vincoli sulle operazioni dei processi interagenti.

Nella cooperazione, la sincronizzazione può essere utile per imporre un particolare ordine cronologico alle azioni eseguite dai processi interagenti; nella competizione, per garantire la mutua esclusione dei processi nell'accesso alla risorsa condivisa.
In caso di condivisione di risorse (variabili) può essere necessario impedire accessi concorrenti alla stessa risorsa (variabile).
Si definisce allora sezione critica, la sequenza di istruzioni mediante le quali un processo può aggiornare variabili condivise. Mediante la mutua esclusione, ogni processo segue le proprie sezioni critiche in modo mutuamente esclusivo rispetto agli altri processi.
Si instaura un comportamento tipo produttore-consumatore, e solo quando la risorsa è offerta dall'uno, questa può essere reperita in maniera esclusiva dall'altro. E' importante evitare i casi di blocco critico o deadlock che si verifica se ogni processo dell'insieme è in attesa di un evento che può essere causato solo da un altro processo dell'insieme.

4.6 Thread

Un thread, talvolta chiamato processo leggero, è l'unità di base dell'uso della CPU e comprende un identificatore di thread (ID), un contatore di programma, un insieme d i registri, e una pila (stack). Condivide con gli altri thread che appartengono allo stesso processo la sezione del codice, la sezione dei dati e altre risorse di sistema, come i file aperti e i segnali.

4.6.1 Caratteristiche dei thread

Essi eseguono all'interno del contesto di esecuzione di un unico processo. Non hanno uno spazio di indirizzamento riservato: tutti i thread appartenenti allo stesso processo condividono lo stesso spazio di indirizzamento; in più hanno execution stack e program counter privati. Un processo tradizionale, chiamato processo pesante è composto da un solo thread. Un processo multithread invece ha un context-switch meno oneroso, una comunicazione più semplice, ma ha problemi di sincronizzazione più rilevanti e frequenti.

4.6.2 Multithreading in Java

Ogni esecuzione della macchina virtuale dà origine ad un processo, e tutto quello che viene mandato in esecuzione da una macchina virtuale dà origine ad un thread.
Le specifiche della JVM stabiliscono che una VM gestisca i thread secondo uno scheduling preemptive.
Quindi non c'è garanzia che sia implementata una gestione in time slicing; inoltre, c'è totale liberà sulle modalità di mappatura tra thread java e thread del S.O.

4.6.3 Sincronizzazione dei thread

Differenti thread che fanno parte della stessa applicazione java condividono lo stesso spazio di memoria.
E' possibile che più thread accedano contemporaneamente allo stesso metodo o alla stessa sezione di codice di un oggetto; si evince che occorrono meccanismi di sincronizzazione.
JVM supporta la sincronizzazione nell'accesso a risorse tramite la keyword ***synchronized***.
Si utilizzano dei lock per ottenere la mutua esclusione.
Synchronized può essere utilizzato con scope diversi su:
°**singolo metodo**: lock sull'oggetto su cui viene invocato il metodo
°**blocco di istruzioni** (sezione critica): lock su un oggetto specificato come parametro.

A ogni oggetto java è automaticamente associato un *lock*. Per accedere a un metodo o a una sezione synchronized, un thread deve prima acquisire il lock dell'oggetto di cui si invoca il metodo o su cui è sincronizzata la sezione critica. Il lock è automaticamente rilasciato quando il thread esce dal metodo o dalla sezione synchronized, o se viene interrotto da un'eccezione. Un thread che non riesce ad acquisire un lock rimane sospeso sulla richiesta della risorsa fino a che il lock non è disponibile.
Ogni oggetto java fornisce metodi di sincronizzazione:

wait(): blocca l'esecuzione del thread invocante in attesa che un altro thread invochi i metodi notify() o notifyAll() per quell'oggetto. Il thread invocante deve essere in possesso del lock sull'oggetto; il suo blocco avviene dopo aver rilasciato il lock.

Notify(): risveglia un unico thread in attesa dell'oggetto in questione. Se più thread sono in attesa, la scelta avviene in maniera arbitraria, dipendente dall'implementazione della macchina virtuale Java. Il thread risvegliato compete con ogni altro thread per ottenere la risorsa protetta.

NotifyAll(): esattamente come notify(), ma risveglia tutti i thread in attesa per quell'oggetto in questione

Conclusioni

In questo capitolo si è cercato di mettere in risalto le scelte che sono state effettuate nella realizzazione del progetto, avendo avuto cura di evidenziare anche i passi implementativi per una migliore esplicazione di quanto descritto, senza trascurare anche i richiami teorici.

Capitolo 5

Wireless Toolkit e test

5.1 Impostazione della fase di test

Per effettuare le varie prove riguardanti l'accertamento del funzionamento dell'applicazione, ma anche per le fasi di compilazione e debug, si sono sfruttate le funzionalità messe a disposizione dell'emulatore Wireless Toolkit. Grazie a questo, si è potuta svolgere la fase di test del progetto, ma si è potuto prendere anche atto dei tempi necessari di realizzazione della completa risposta e dunque del completo svolgimento del flusso multimediale.
Prima di evidenziare i tratti che si possono sottolineare dalla fase di test, è bene effettuare una descrizione dello strumento che è stato adottato.

5.2 Il Wireless Toolkit

L'emulatore provvede un ambiente di esecuzione e un software applicativo di gestione per il Wireless Toolkit.

Il Wireless toolkit J2ME supporta lo sviluppo delle applicazioni Java che eseguono su dispositivi con un Mobile Information Device Profile (MIDP), come i telefoni cellulari. In più, il Wireless Toolkit supporta anche lo sviluppo di applicazioni Wireless MessagingAPI (WMA) e Mobile Media API (MMAPI) dispongono.
La Ktoolbar, inclusa con il Wireless Toolkit J2ME, è un piccolo ambiente di sviluppo con una interfaccia grafica (GUI) per la compilazione e l'esecuzione delle applicazioni MIDlets.
Un IDE compatibile con il Wireless Toolkit prevede molte funzionalità, per esempio, se si usa il Wireless dentro un IDE, è possibile editare, compilare, ed eseguire le applicazioni MIDlets, tutte dallo stesso ambiente.
Il Wireless Toolkit provvede un' implementazione del MIDP insieme ad un emulatore che può essere utilizzato per la visualizzazione di un

risultato su un display. Non è comunque un ambiente di sviluppo completo, perché non fornisce un editor integrato che permette di realizzare, vedere, e modificare il codice sorgente. Di conseguenza, se il Wireless Toolkit dovesse essere utilizzato come parte di un ciclo completo di sviluppo, occorrerebbe un editor o un IDE per gestire il codice sorgente.

5.2.1 Configurazione dell'emulatore

Il primo passo da compiere quando si usa il Wireless Toolkit è quello di creare un progetto, che gestisce il codice sorgente, le classi, e le risorse corrispondenti ad un Midlet suite. Per fare questo occorre aprire la Ktoolbar e premere su NewProject per aprire la finestra di dialogo, come è mostrato sotto.

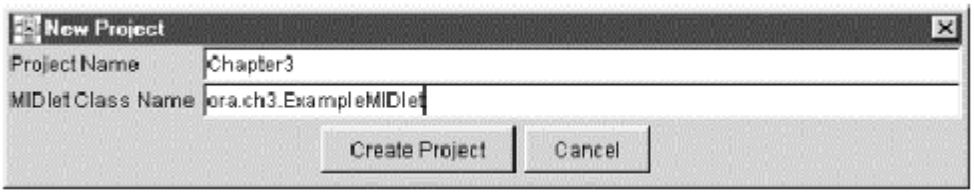

Figura 19: creazione di un progetto con il Wireless Toolkit

Un progetto Ktoolbar è associato ad un MIDlet suite. Il progetto contiene il sorgente della suite, la risorsa, i file binari, il JAD e il file manifest che contiene gli attributi della suite. I file del progetto vengono salvati in una sottodirectory della directory d'installazione del Wireless Toolkit che viene creata al momento dell'installazione dell'emulatore stesso. La tabella sotto mostra come i file sono organizzati:

Directory	Description
{*j2mewtk.dir*}\apps\{*project.name*}	Contains all source, resource, and binary files of the project
{*j2mewtk.dir*}\apps\{*project.name*}\bin	Contains the JAR, JAD, and unpacked manifest files.
{*j2mewtk.dir*}\apps\{*project.name*}\lib	Contains external class libraries, in JAR or ZIP format for a specific project.
{*j2mewtk.dir*}\apps\{*project.name*}\res	Contains all the resource files.
{*j2mewtk.dir*}\apps\{*project.name*}\src	Contains all the source files.
{*j2mewtk.dir*}\apps\lib	Contains external class libraries, in JAR or ZIP format for all KToolBar projects.

Figura 20: organizzazione dei file

Dopo aver premuto il bottone CreateProject, il Wireless Toolkit apre un'altra finestra di dialogo (Settings);

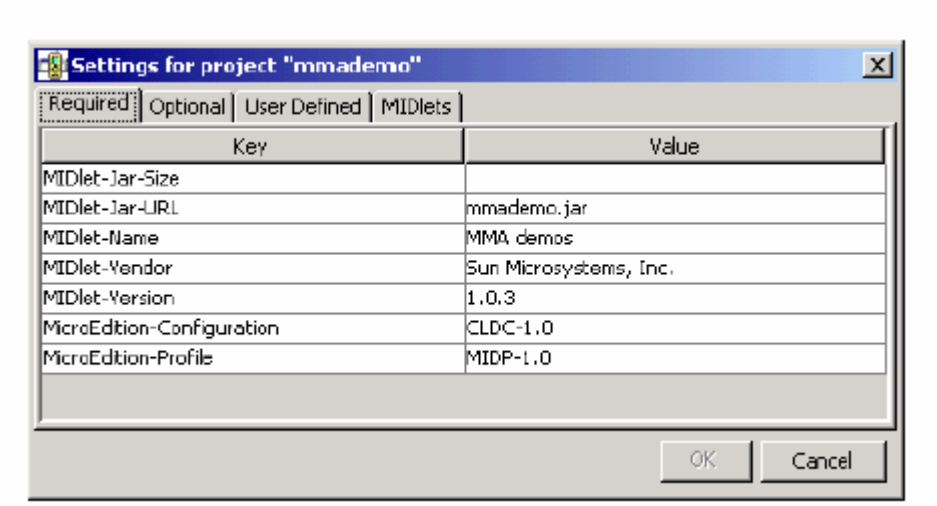

Figura 21: finestra di dialogo Project Settings

essa contiene un set di tabelle, tra le quali la tabella *Required*, che contiene delle celle che permettono di provvedere agli attributi usati per generare il manifest per il JAR file e il JAD file. E' possibile editare questi attributi cliccando sopra le celle che si vogliono cambiare e inserendo il nuovo valore. Molti dei valori specificati possono essere

usati senza nessuna modifica. Il campo MIDlet-Name (che indica il nome attuale usato per la suite MIDlet) fa match con il nome del progetto, e il nome del file JAR che sarà creato dopo deriva dal nome del progetto.
Un'altra tabella importante è la *MIDlets* che contiene delle celle che permettono di settare il nome del MIDlet e la classe dalla quale viene eseguita.
E'possibile modificare anche degli attributi definiti dall'utente e questo è possibile farlo mediante la tabella *User-defined*.

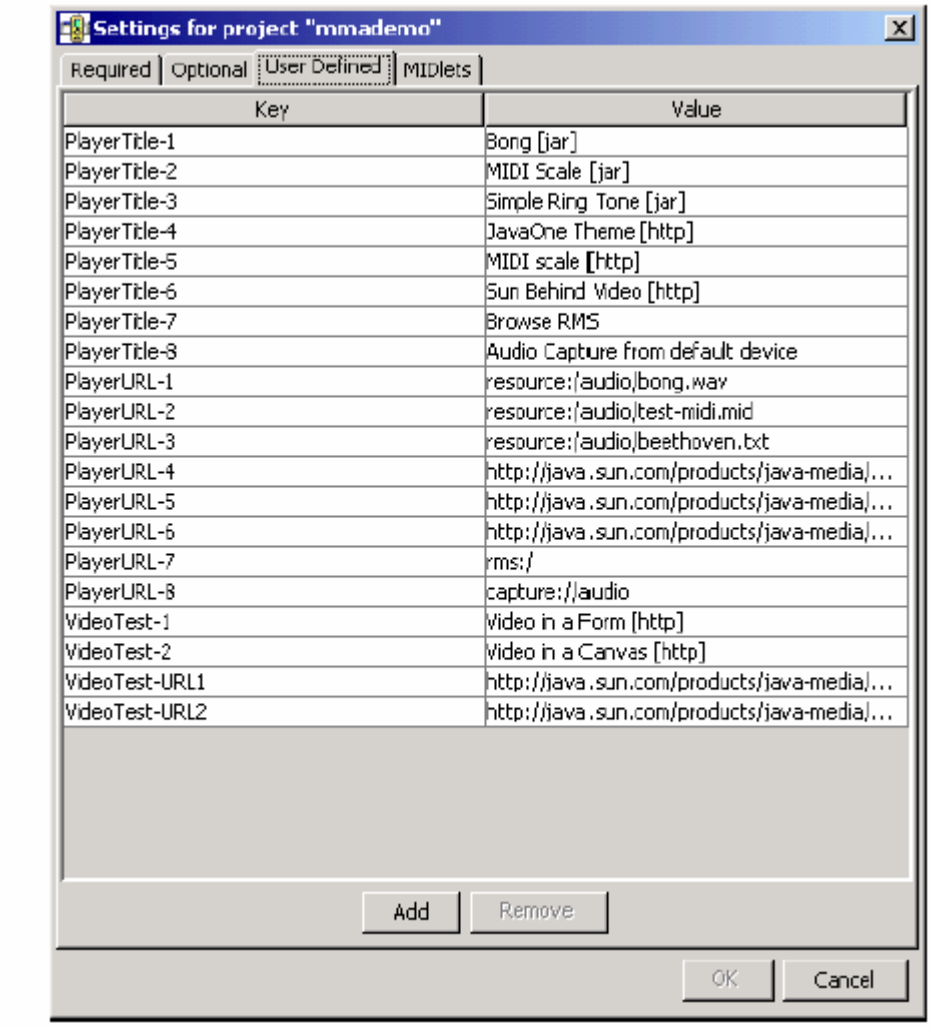

Figura 22: finestra User-Defined Attributes

Si possono aggiungere od eliminare gli attributi relativi al push registry mediante la relativa tabella *PushRegistry*:

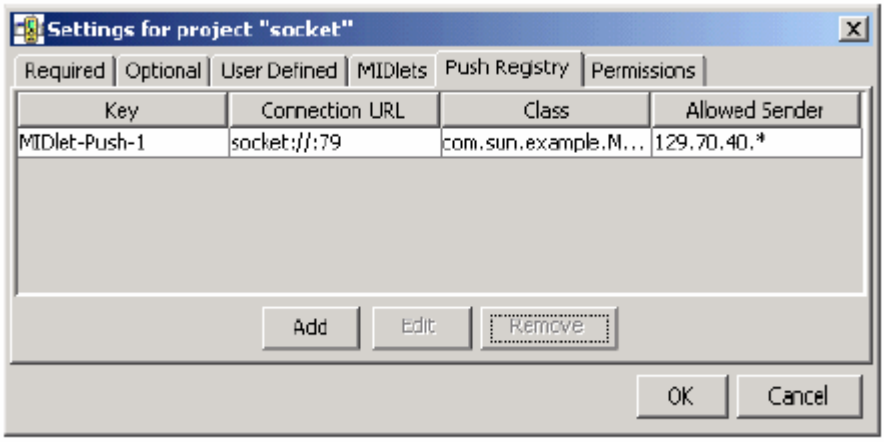

Figura 23: esempio di un Push Registry

Nella cella relativa al Connection URL occorre specificare una stringa che identifica il protocollo di connessione e il numero di porta.
Nella cella relativa alla Class occorre indicare invece il nome della classe del MIDlet. Se la MIDlet data appare più volte nella suite, viene usato il primo entry. Il MIDlet deve essere registrato in un file JAD con un record che riporta la chiave del MIDlet (MIDlet -<n>).
Nella cella relativa all'AllowedSender occorre indicare un valido sender che può lanciare il MIDlet. Se il valore riportato è "*", vuol dire che sono accettate le connessioni provenienti da qualunque sorgente.
Per una suite MIDlet con più di un MIDlet, occorre aggiungere una riga in più per ogni MIDlet.
Il passo successivo consiste nel localizzare il codice sorgente per il Midlet dove il Wireless Toolkit può accedere ad esso. La maggior parte degli IDE permette di scegliere dove tenere i file del progetto, invece il Wireless Toolkit usa un filesystem per ogni progetto, basato su una directory esistente da quando il Wireless Toolkit viene installato. Il nome della directory di un progetto deriva dal nome dato al progetto stesso all'atto della sua creazione. Se, per esempio, il Wireless Toolkit viene installato sotto la directory C:\ *J2MEWTK,* tutti i file del nuovo progetto che è stato chiamato "pippo", verranno salvati sotto la directory *c:\J2MEWTK\apps\pippo.*

Quando il nuovo progetto viene creato, il toolkit crea le seguenti cartelle:
src: in questa cartella vengono mantenuti i file del codice sorgente per il MIDlet.
res: in questa cartella vengono salvate le risorse richieste per il MIDlet, come video, icons.
bin: mantiene il file manifest, i JAR e i JAD.

5.2.1.1 Api Permissions

Affinchè una suite MIDlet possa operare, occorre che acceda a determinate API protette. Per ottenere il permesso di accesso a queste API è necessario effettuare una richiesta.
E' possibile settare gli attributi `MIDlet-Permissions e MIDlet-Permissions-Opt` dal pannello Permission del box di dialogo Settings.
Per specificare a quali api la corrente suite di MIDlets può accedere:
1. Scegliere Project -> Settings e cliccare sulla tabella Permissions.
2. Cliccare Add per ulteriori richieste (MIDlet-Permissions) o optionali (MIDlet- Permissions-opt).

5.3 Limitazioni nell'uso delle MMAPI nel WirelessToolkit

Occorre tener presente che il Wireless Toolkit ha le seguenti limitazioni e che ogni dispositivo riscontrerà le stesse ma in maniera diversa:

1.Un'applicazione può creare diversi Player, ma può realizzare solo un MIDI Player o una sequenza di toni. Ogni tentativo nel realizzare un secondo Player di un altro tipo lancerà un MediaException. In altre parole, può essere usato un Player alla volta.

2. Il Toolkit permette un certo numero di chiamate simultanee playTone() per l'esecuzione di una singola nota, ma alla fine solo quattro saranno veramente al lavoro.

3. Il numero di player video è limitato dalla dimensione dell'heap, come specificato nella Ktoolbar; **Edit -> Preferences -> Storage**.

E’ da notare anche che il MIDP limita il numero di connessioni aperte simultaneamente a quattro.

5.4 Preferenze dell’emulatore

Per un buon utilizzo dell’emulatore, occorre setttare delle opzioni come la dimensione dello storage e dell’heap size. Questo viene effettuato per specificare la memoria disponibile per le applicazioni che dovranno essere eseguite su un dispositivo che supporta le MMAPI.
Dalla Ktoolbar basta cliccare su Edit -> Preferences. A questo punto, occorre specificare un valore nel campo Heap Size (in kilobytes). Il valore di default è 4000KB. I valori ammessi vanno da 32KB a 64000KB. Il valore minimo dell’heap dipende dal tipo di MIDlet che si sta eseguendo. La tavella seguente riporta le dimensioni dell’heap richieste per un emulatore MMAPI e per le applicazioni che si intende eseguire.

Minimum Heap Size Requirements for MMAPI MIDlet

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`)

Figura 24: minimo heap size richiesto per una Midlet

Setting the Web Proxy

Se si vuole eseguire un’applicazione Java che richiede una connessione web, e se le connessioni possono essere realizzate solo mediante un server proxy (quando,ad esempio, un server web è sull’altra parte di una

firewall), occorre configurare l'emulatore con delle informazioni riguardanti un server proxy.
Occorre in tal caso specificare il nome del server proxy http e il suo numero di porta, nei rispettivi campi di testo.
Setting Network Speed Parameters

Molto spesso la performance di un'applicazione è data dalla velocità che caratterizza la rete. Per vedere come variano le caratteristiche di un'applicazione in base alla velocità o alla larghezza di banda utilizzata, è possibile modificare i parametri. Questo è possibile selezionando i parametri desiderati nella tabella di Performance relativa alle Preferences.

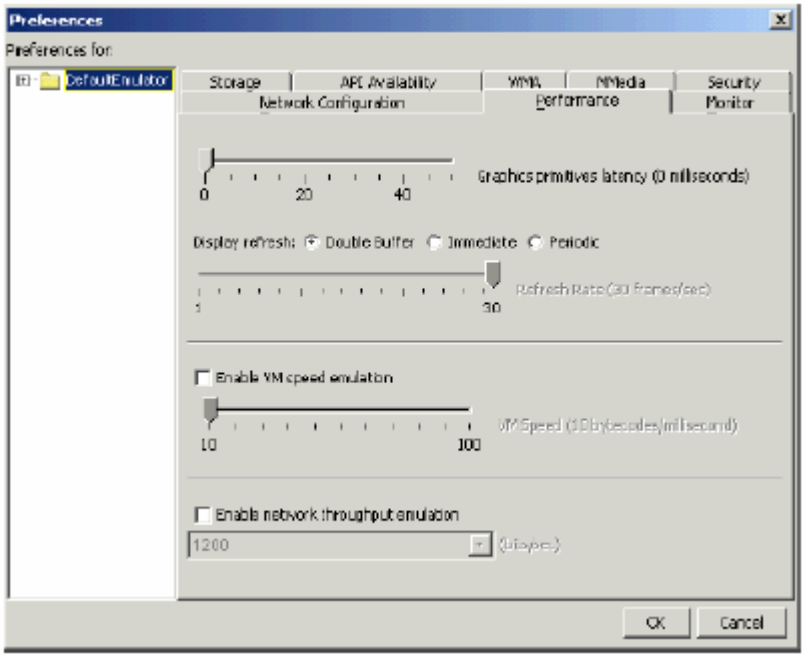

Fig 25: Performance Settings

Ciccando su 'Enable network throughput wmulation', è possible settare I valori di larghezza di banda con i quali si vuole lavorare.

5.5 Esecuzione di un MIDlet

A questo punto, il file JAR non è ancora stato creato, ma si può testare la suite MIDlet selezionando un appropriato target sulla finestra principale della Ktoolbar e premendo il bottone della Run.
Questo carica le classi MIDlet, le sue risorse, le librerie associate da classes, res, e lib.
Premendo la Run, l'emulatore inizia e rappresenta la lista di MIDlets presenti nella suite.

The Wireless Toolkit emulator
Figura 26: emulatore Wireless Toolkit

Quando una suite MIDlet viene caricata, il gestore dell'applicazione mostra una lista delle midlet che sono presenti e permette di selezionarne una ed iniziare la sua esecuzione.
Potrebbe essere pericoloso per un dispositivo lanciare una MIDlet automaticamente.
Non è ovvio pensare su quali basi una decisione verrà fatta, cioè da quando l'utente vedrà solo i nomi dei MIDlet, dovrà confermare l'inizio dell'esecuzione del MIDlet. In questo caso, il dispositivo rappresenta il nome ddel MIDlet e la sua icona presa dagli attributi (MIDlet-1) nel file manifest. Il dispositivo non è obbligato a raffigurare l'icona, potrebbe usare la sua invece che quella specificata nel file manifest.
Quando viene lanciato un MIDlet, il Wireless Toolkit compila il codice sorgente con l'opzione di salvare le informazioni di debugging in un file, ma ancora non viene creato il file JAR.

Per creare il file JAR, occorre selezionare la preferenza Package dal menu Project. Così facendo, tutte le classi vengono ricreate senza debugging, riducendo le dimensione del file delle classi; una misura intesa per mantenere il tempo richiesto per scaricare il JAR su un dispositivo cellulare il più breve possibile.
Il file JAR può essere usato con il file JAD per distribuire la suite MIDlet per le installazioni in dispositivi su una rete.

5.6 Fase di test

Per analizzare le caratteristiche del prototipo realizzato sono stati eseguiti alcuni test. Questi test hanno anche lo scopo di mettere in risalto lo stato dell'arte della tecnologia Java embedded nel settore della riproduzione di contenuti multimediali. E' stato sottolineato lo strumento utilizzato per effettuare le simulazioni e fino al paragrafo precedente ne sono state sottolineate alcune caratteristiche.
Mediante l'emulatore Wireless Toolkit si è cercato di ottenere dei risultati che potessero rispecchiare il più possibile quelli ottenibili dall'uso di un vero dispositivo portatile. Sicuramente non si può parlare di ottima risoluzione delle immagini o possibilità di miglioramento, ma quello che sicuramente interessa e si vuole evidenziare sono i tempi necessari al completo download della risorsa.

Il test consiste nello scaricare un presentazione multimediale da un indirizzo http. La presentazione in oggetto è un file video di 660 KB la cui durata è di 21 secondi, per essere riproducibile dal package MMAPI deve essere codificato in Mpeg, è a colori ed ha una risoluzione di 180x177. Le prove sono state fatte sfruttando la possibilità del *Wireless Toolkit* di simulare le velocità ottenibili con alcuni standard di trasmissione.

Il test è stato effettuato considerando che la presentazione multimediale presente sul server sia disponibile in cinque parti.
Dopo che il client avanza la sua richiesta tra le possibili, si crea una connessione di tipo Client-Server tra l'agente che possiede tale file ed il dispositivo mobile.Il test è stato effettuato per evidenziare i tempi di scaricamento di ciascun pezzo della presentazione e per poi metterlo a

Figura 27: interfaccia grafica del videoPlayer

confronto con i relativi tempi di presentazione. Avendo a disposizione una presentazione dalla durata di 21 secondi, ogni pezzo ha la durata di 4,2 secondi. Se il tempo di presentazione è più piccolo del tempo di scaricamento di ciascuna banda, allora si garantisce una buona visualizzazione anche di fronte a possibili disconnessioni. Il test è stato realizzato considerando diverse larghezze di banda, evidenziando il fatto che con una banda più largha è possibile scaricare in minor tempo e si riesce ad ottenere anche un miglior risultato nella rappresentazione su display.

Si è tenuto conto della banda relativa a dispositivi Bluetooth che hanno una banda di 721 Kbps e, come mostra la tavella, si evince che questa banda è sufficiente per ottenere i risultati sperati e cioè che i tempi di scaricamento sono più veloci dell'effettiva visualizzazione di ciascun pezzo. Utilizzando invece bande relative a dispositivi GPRS e GSM, la questione diventa un po' più disagevole in quanto i tempi di scaricamento divengono più lenti, con il rischio che la visualizzazione del filmato avvenga in maniera spezzata. Utilizzando dunque una banda abbastanza largha si ottengono dei risultati soddisfacenti anche di fronte a possibili cadute di connessione, in quanto si garantisce una buona visualizzazione del filmato che è disponibile in locale.

Standard	Velocità	Tempo di download totale	Tempo di download per pezzo 1/5	Tempo di presentazione per pezzo
Bluetooth	721 Kbs	7.3 s	1.46 s	4.2 s
GPRS	33.6 Kbs	157 s	31.4 s	4.2 s
GSM	9.6 Kbs	550 s	110 s	4.2 s

Figura 28: risultati del test di download

Per settare i valori relativi alle larghezze di banda desiderati, fare riferimento alla sezione Setting Network Speed Parameters del paragrafo 5.4.
I tempi della fase che prevede l'instaurarsi della connessione sono abbastanza brevi, seguita dalla fase che vede la trasmissione e la realizzazione dei player. Quest'ultima fase vede il trasferimento di tutte le parti mentre è in esecuzione l'ultima che è stata ricevuta dal dispositivo; appena arriva la prima si effettua la creazione del primo player. Mentre è in esecuzione la prima parte, si effettua la creazione del secondo player di modo che alla terminazione del primo, possa entrare subito in esecuzione per effettuare un merging e una rapida successione delle parti. Il ragionamento vale per ogni pezzo ancora disponibile a costituire il media. Il test prevede anche una valutazione del tempo impiegato nel passaggio da un player all'altro, ovvero nel passaggio da un pezzo all'altro, per valutare quanto possa essere percepita la cucitura da parte dell'utente. La fase di test ha mostrato che il tempo impiegato è minimale, dell'ordine di 1 millisecondo, ma percettibile all'occhio umano.

Il grande vantaggio di questa realizzazione sta nella possibilità di rappresentare il media senza possederlo tutto in locale, perché altrimenti i tempi di attesa sarebbero stati più lunghi proprio per l'eventuale caricamento di tutta la risorsa sul dispositivo. Questo avrebbe reso il servizio ancora meno qualitativo, tenendo conto anche delle limitazioni della memoria del dispositivo e ne avrebbe rallentato anche l'efficienza rappresentativa.
I risultati ottenuti sono positivi per quanto riguarda la scelta appena descritta e ovviamente adottata, un po' meno per i tempi di realizzazione ottenuti con l'uso di bande non abbastanza sufficienti per rendere il trasferimento così veloce da avere tempi minori di quelli di visualizzazione .
Dunque, l'approccio adottato e l'ausilio dell'emulatore sono stati efficienti per la piena realizzazione del progetto.

Conclusioni

Il lavoro svolto in questa tesi ha avuto come obiettivo fondamentale la realizzazione di un progetto che permettesse la fruizione di materiale multimediale da dispositivi dalle capacità limitate, come i moderni cellulari. Questo ha portato ad approfondire prima alcuni concetti relativi alla J2ME, ampiamente utilizzata per la realizzazione di un client installabile su cellulare che gestisse le funzionalità richieste, poi ad esplorare la libreria JMF impiegata per trattare opportunamente il materiale multimediale sul server. L'applicazione sviluppata è stata poi testata grazie all'uso del Wireless Toolkit e di uno standard server web

opportunamente esteso. Il lavoro è stato svolto in maniera graduale e con accrescimenti che di volta in volta hanno portato ad approfondire e risolvere le problematiche che venivano incontrate. Si è giunti ad estendere il progetto in modo tale da poter gestire fino ad n pezzi costituenti un'applicazione multimediale con il supporto e la sola esecuzione di due soli player; la scelta di due soli player è legata alla possibilità di gestire in maniera più veloce il merging delle varie parti e di rendere la realizzazione del progetto il più flessibile possibile. La fase di testing è stata effettuata fin dall'inizio per la verifica di ogni risultato che si voleva ottenere. Il testing finale è stato effettuato per vedere i tempi di scaricamento di ciascun pezzo costituente la presentazione multimediale; con il tempo di scaricamento ottenuto si effettua il confronto con il tempo di presentazione. Utilizzando una banda come quella di un dispositivo Bluetooth, si ha che il tempo di scaricamento è minore del tempo di presentazione di ciascun pezzo e, grazie a questo, ogni pezzo ricevuto può essere visualizzato senza interruzione, a differenza dei risultati ottenuti con l'utilizzo di una banda minore che permette un trasferimento minore di dati; questo caso non permette di ottenere un risultato soddisfacente nella visualizzazione della presentazione in quanto per ogni pezzo visualizzato su display si avrà un'attesa per lo scaricamento del successivo pezzo deve termina alcuni istanti dopo l'avvenuta rappresentazione del pezzo precedente. Occorre tenere in considerazione gli strumenti che sono stati impiegati e il fatto che con l'impiego di un emulatore si è cercato di ottenere dei risultati che approssimano il più possibile quelli ottenibili da un dispositivo reale.

L'applicazione permette di effettuare un download della presentazione multimediale senza doverla memorizzare per intero sul xlient prima di poterla visualizzare. La corretta successione delle varie parti che arrivano sul client e la conseguente visualizzazione di ciascuna di esse permette di dare una spiegazione a ciò che chiamiamo rendering della risorsa.

La spiegazione di tale scelta sta appunto nel cercare di realizzare l'obiettivo che si è posto nell'affrontare questa tesi, e cioè la possibilità di venire incontro ad una utenza che richiede di poter reperire le informazioni su un dispositivo wireless. Questo richiede la conoscenza della risorsa ma si cerca di non vincolare la stessa ad un dispositivo in

particolare e cioè senza il vincolo della memorizzazione prima dell'effettiva visualizzazione.
Il progetto è stato svolto con l'ausilio di standard come Java, il che permette di avere una certa portabilità e si è fatto uso di un web server standard, per poter realizzare un'applicazione lato client con le dovute funzionalità.
Il progetto sviluppato è stato realizzato arrivando a definire gli obiettivi che si sono stati posti fin dall'inizio ma è possibile migliorare ancora l'applicazione mediante degli sviluppi futuri come una migliore gestione della qualità nel passaggio da una traccia alla successiva, di modo che l'utente non si accorga della cucitura che avviene da un pezzo al successivo ed abbia come risultato finale una visualizzazione scorrevole e fluida.

Ringraziamenti

I miei ringraziamenti vanno al prof. Antonio Corradi per avermi dato l'opportunità di svolgere il lavoro della seguente tesi e all'ing. Luca Foschini che mi ha guidato con le sue indicazioni e i suoi consigli, fiduciosa nella loro puntualità e disponibilità.
Un ringraziamento particolare va alla mia famiglia che nonostante la lontananza mi sono stati sempre vicini, in particolar modo nei momenti in cui non credevo nelle mie capacità e nelle mie forze. Mi hanno dato la forza di essere tenace e di cercare in me quel poco di coraggio che mi avrebbe fatto andare avanti. Un grazie va a tutti i miei amici con i quali ho condiviso in questi anni accademici tanti momenti felici e sono stati il mio scudo nei momenti più bui che mi sarebbero potuti capitare. Con loro ho veramente capito tante cose e ho ritrovato il vigore, la forza di andare avanti e non nascondermi perché almeno di loro potevo fidarmi. Grazie a: Marco, Davide, Luca, Campel, Berga, Bambo, Paolo, Claudia, Simone.
Un ringraziamento va in particolare a: Giuseppe, Fede, Nicolò, Giovanni, Andrea, Giovanni, Serena, Alessio che in questi giorni di "cauto" stress mi hanno fatto trascorrere delle magnifiche giornate in loro compagnia, circondata dall'affetto e dal loro sostegno, inconsapevoli tante volte di avermi aiutata anche quando non sembrava io fossi triste. Grazie anche alle nuove amiche della mia palestra, in particolare ad Ilaria e Piera con le quali ho trascorso piacevoli ore di allenamento ma anche di svago.
Grazie anche a tutti quelli che ho dimenticato e a tutti coloro che inconsapevolmente mi hanno aiutato e come dice un mio caro amico:"L'importante non è cadere ma rialzarsi ed andare avanti".

Riferimenti bibliografici

Betti, “”, tesi di laurea presso l’Università di Bologna, Facoltà di Ingegneria, a.a. 2000/2001.

Borelli, “”, tesi di laurea presso l’Università di Bologna, Facoltà di Ingegneria, a.a. 2000/2001.

Foschini, “”, tesi di laurea presso l’Università di Bologna, Facoltà di Ingegneria, a.a. 2002/03.

Java Media FrameWork API Guide, novembre 1999.

Kim Topley, “J2ME in a Nutshell”, marzo 2002.

Nokia, “Birth of Mobile Java Multimedia”.

Sun Microsystems, UserGuideWirelessJ2ME.

Sun Microsystems, “Think small with J2ME”.

Sun Microsystems, “J2ME Mobile Media Api bringing time-based multimedia to consumer device”.