UNIVERSITÀ DEGLI STUDI DI TRENTO

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea triennale in Informatica

Elaborato Finale

LEZIONI SUL TELEFONINO: PORTING IN AMBIENTE SYMBIAN

Relatore:
prof. Marco Ronchetti

Laureando:
Jovan Stevovic

Anno Accademico 2006-2007

1. Introduzione

1.1. Premessa

Al giorno d'oggi i computer hanno un ruolo dominante. Sono presenti ovunque: sia nella forma più riconoscibile quale personal computer o palmari, sia nelle altre forme dove sono parte integrante di sistemi complessi come ad esempio automobili, console per videogiochi e persino nei frigoriferi e lavatrici. Ciascuno di noi porta sempre con se un piccolo dispositivo con capacità di calcolo maggiore di quella del computer che guidò Apollo e portò l'uomo sulla Luna: questi dispositivi sono i cellulari. Da recenti studi [THO][TRI] su un campione di più di 1000 studenti in tre diversi paesi (Italia, Bulgaria, Giappone) è emerso che tutti posseggono almeno un telefonino mentre meno del 20% possiede un palmare. Molti dei telefonini di ultima generazione sono dei veri dispositivi di calcolo: hanno un vero sistema operativo (ad es. Symbian[1]) e possono essere programmati (ad es. Java Micro Edition[2]). La quantità di memoria disponibile è abbastanza elevata e possono contenere files di grandi dimensioni. Nei casi estremi come ad es. Apple iPhone[3] o vari Nokia N-Series[4] possono avere fino a 8 GigaBytes di memoria ed è possibile eseguire la shell di Linux. Posseggono inoltre vari tipi di connettività da SMS a Bluetooth e hanno accesso totale ad intenet tramite GPRS/EDGE/UMTS o via wi-fi.

E' perciò interessante analizzare come i cellulari possano essere utilizzati da sistemi di mobile learning. Essendo la definizione di mobile learning molto vasta, includendo sia la mobilità dello studente

[1] http://www.symbian.com/
[2] http://java.sun.com/javame/index.jsp
[3] http://www.apple.com/iphone/
[4] http://www.nseries.com/index.html

che quella del dispositivo utilizzato, noi indicheremo con questo termine il porting di applicazioni per l'e-learning su dispositivi mobili.
Recentemente nelle e-community c'è stata un'esplosione d'interesse per il podcasting[5] che permette la condivisione di dati quali tracce mp3 sopratutto sugli iPod[6] della Apple ma anche sui telefoni cellulari. Solo le versioni più costose degli iPod, però, possono riprodurre immagini e video. Molti dei telefonini attualmente in commercio possono riprodurre tracce mp3 ma rispetto agli iPod incorporano al loro interno delle fotocamere e possono sia riprodurre che registrare immagini e video.
Questi fatti hanno ispirato e motivato questo lavoro nel quale si è cercato di implementare un versione estesa di podcasting sui telefonini considerando sia l'audio che informazioni video.

1.2. Panoramica sul lavoro svolto

Utilizzare il telefonino come mezzo di e-learning è un opportunità già studiata da molti. Sono stati approfonditi diversi metodi come ad esempio utilizzare gli SMS per comunicazioni tra studente e docente o per lo scambio di informazioni multimediali. Con i nuovi dispositivi che si trovano attualmente in commercio, le potenzialità sono cresciute notevolmente. Sui nuovi telefonini è possibile facilmente riprodurre immagini, audio e video lunghi anche qualche ora. Il sistema L.O.D.E. (Lectures On-DEmand) [LODE] è un progetto nato per consentire la diffusione di lezioni on-line combinando insieme audio, video, presentazioni PowerPoint sincronizzate (immagini ad alta risoluzione), possibilità di aggiungere nuovi materiali e la possibilità di navigare attraverso la lezione. In

[5] http://it.wikipedia.org/wiki/Podcasting
[6] http://www.apple.com

questo lavoro si è cercato di portare il sistema LODE nel mondo Mobile. E' stato utilizzato come ambiente di sviluppo JavaME soprattutto per motivi di portabilità e perché al giorno d'oggi la KVM (Java Virtual Machine per dispositivi mobili) è presente in quasi tutti i telefonini ed è stato utilizzato in particolare un modello di telefono avente sistema operativo Symbian OS 8.0a, S60 2nd Edition FP 2. E' stato creato inoltre un sistema di conversione completamente automatico, che permette di produrre lezioni compatibili con il programma per telefonini partendo dalle lezioni del sistema LODE. Le varie caratteristiche e programmi utilizzati verranno spiegati dettagliatamente nei capitoli successivi.

2. Architettura mobile

2.1. Considerazione Hardware

Il progetto è stato sviluppato in parte su un Nokia 6680[7] e in parte su un Nokia N70[8]. Questi due dispositivi hanno caratteristiche hardware pressoché uguali e sopratutto posseggono le stesse librerie JavaME. Nella seguente tabella sono elencate le principali caratteristiche hardware dei dispositivi utilizzati per lo sviluppo e altri dispositivi attualmente in commercio.

	Anno di prod.	**Sistema operativo**	**Proc-essore**	**Mem. Ram**	**Mem. Inter-na**	**Mem. Espan-dibile**	**Risol-uzione Schermo**	**Dim. Sch-ermo**
Nokia 6680	2005	Symbian OS 8.0a, S60 2nd Edition FP 2	220 (MHZ)	8 (MB)	10 (MB)	2(GB)	176 x 208 (pixels)	2.1″
Nokia N70	2005	Symbian OS 8.0a, S60 2nd Edition FP 2	220 (MHZ)	32 (MB)	22 (MB)	2(GB)	176 x 208 (pixels)	2.1″
Nokia N95 8GB	2007	Symbian OS 9.2, S60 3rd Edition, FP 1	332 (MHZ)	84 (MB)	8 (GB)	4(GB)	240 x 320 (pixels)	2.8″

[7] http://www.nokia.it/A4190102
[8] http://www.nokia.it/A4346090

Apple iPhone	2007	iPhone's OS X	620 (MHZ)	128 (MB)	16 (GB)	-	320 x 480 (pixels)	3.5”

Tabella 2.1.1: *Caratteristiche Hardware dei dispositive utilizzat e attualmente disponibili.*

Alcune delle caratteristiche hanno influenzato pesantemente lo sviluppo e le scelte implementative. Quelle di maggior impatto sono state la quantità di memoria RAM e la risoluzione dello schermo. Per quanto riguarda la memoria fisica, grazie a delle schede di memoria MMC si è potuto espandere la memoria già presente in modo da aggirare ogni limite. Da quando il progetto è stato iniziato, sono usciti in commercio dispositivi con caratteristiche più che sufficienti per un efficiente utilizzo e ulteriore incremento di funzionalità offerte. Come si può notare la quantità di memoria presente negli ultimi modelli è sufficiente per mantenere l’intera presentazione che si sta consultando nella memoria RAM, ovviando così a ogni problema riscontrato (ad es. permessi, lettura sequenziale di files ecc.). Le soluzioni proposte saranno spiegate nei capitoli successivi. In seguito è elencata un’analisi approfondita di ogni caratteristica hardware:

- Processore: la velocità di clock del processore dei dispositivi utilizzati è l’unica caratteristica pienamente soddisfacente. E’ sufficiente sia nella riproduzione degli audio che nella visualizzazione delle immagini. L’unico limite riscontrato è la velocità con cui viene effettuato lo zoom-in e zoom-out delle immagini. C’è da considerare però che l’algoritmo utilizzato eseguiva molti accessi alla memoria RAM e di conseguenza la lentezza dell esecuzione è causata anche da quest’ultimi.
- Memoria RAM: nel primo dispositivo utilizzato era di soli 8 MegaBytes. Ogni presentazione è composta da alcune tracce audio e da immagini e occupa all’incirca 20 MegaBytes di

memoria. Non è possibile, quindi, caricare in memoria interamente ne le immagini ne le tracce audio. La lentezza di accesso ai dati della scheda di memoria (MultiMediaCard[9]) ha comportato l'implementazione di un apposito *thread*[10] che pre-caricava in memoria le parti che sarebbero state visualizzate in un futuro breve. Nel secondo dispositivo utilizzato la quantità di memoria RAM è 32 Mega. Essa è sufficiente per un efficiente implementazione della lettura di immagini e tracce audio. Si era scelto agli inizi di non modificare la struttura del progetto per un fatto di compatibilità con dispositivi che non posseggono un elevato quantitativo di memoria. A causa di altri problemi di compatibilità però, si è scelto di sfruttare tutte le capacità dei nuovi modelli a scapito della compatibilità con quelli antecedenti.

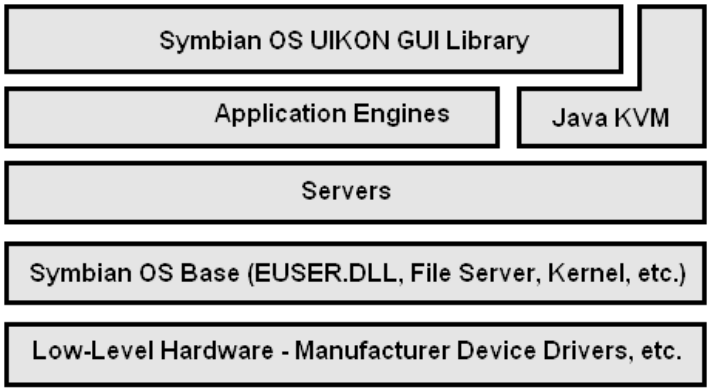
- Memoria interna: non è stato possibile utilizzare la memoria fisica interna dei dispositivi perchè non è sufficiente per contenere le presentazione.
- Memoria espandibile: la memoria fisica è espandibile con delle schede di memoria differenti a seconda del modello. Quelle utilizzate dai dispositivi usati per lo sviluppo sono MMC (MultiMediaCard). Le loro dimensioni possono arrivare fino a 2 GigaBytes e sono più che sufficienti per contenere molte presentazioni.
- Risoluzione schermo: la dimensione dello schermo dei dispositivi non è sufficiente per visualizzare le immagini intere in modo da leggere tutti i dettagli presenti in esse. Di conseguenza sono state implementate delle funzionalità di ingrandimento che consentono di leggere tutti i dettagli. Si può affermare che le dimensioni sono sufficienti per una buona visione dei contenuti con l'ausilio di funzioni di ingrandimento.

[9] http://it.wikipedia.org/wiki/MultiMediaCard
[10] http://it.wikipedia.org/wiki/Processo_%28informatica%29

2.2. Symbian OS

Symbian è un sistema operativo multithreading[11] e multitasking[12] per dispositivi mobili prodotto da Symbian Ltd. Appartiene alle più grandi aziende produttrici di telefonini e palmari (es. Nokia, Sony Ericsson, Samsung ecc). Non è un sistema operativo open source[13] però, a differenza di molti altri sistemi operativi per terminali mobili (es. iPhone's OS X), la documentazione relativa alle API[14] è di dominio pubblico. A differenza di altri sistemi operativi quindi, è possibile produrre applicazioni che sfruttano le API e utilizzano le funzionalità offerte. Possono essere utilizzati diversi linguaggi di programmazione quali C++, OPL, Visual Basic, Python, FlashLite e Java con supporto della JVM. Essi, come accade anche nella scrittura di programmi per computer, offrono delle funzionalità ma hanno anche delle limitazioni. Nel caso del Symbian le limitazioni imposte dal sistema operativo sono di diverso tipo. Alcuni limiti sono imposti dall'hardware dei dispositivi stessi mentre altri sono causati dal scelte progettuali relative alla sicurezza.

[11] http://it.wikipedia.org/wiki/Multithreading
[12] http://it.wikipedia.org/wiki/Multitasking
[13] http://it.wikipedia.org/wiki/Opensource
[14] Application Programming Interface API:
http://it.wikipedia.org/wiki/Application_programming_interface



Struttura del Symbian OS

2.2.1. Sicurezza

La sicurezza nel Symbian OS è un punto molto importante. Sono state adottate molte strategie per limitare la diffusione dei virus e malware[15] già dalla prima uscita sul mercato delle prime versioni del sistema operativo. La principale strategia adottata è basata sulla firma delle applicazioni e sulla restrizione dei privilegi. Dalla versione 9.0 ogni applicazione prodotta da sviluppatori di terze parti deve essere firmata per avere accesso alle funzionalità "sensibili". Esse sono elencate in seguito:

- Accesso alla rete
- Messaging
- Autostart delle applicazioni
- Connettività
- Multimedia
- Lettura o Modifica dati utente
- Attivazione chiamate

[15] http://it.wikipedia.org/wiki/Malware

I modelli utilizzati per lo sviluppo del progetto si basavano sulla versione 8.0 nei quali questo tipo di problema poteva essere risolto autocertificando l'applicazione o dando conferma ad ogni utilizzo delle funzioni sensibili. C'è da considerare che tutti i successivi modelli a quello usato nel progetto utilizzano Symbian 9.0 o superiore. Siccome l'applicazione realizzata fa utilizzo di alcune operazioni "sensibili", non potrebbe funzionare su una versione 9.0. senza un certificato valido. Recentemente però, è stato reso disponibile direttamente dalla Nokia un servizio, destinato a sviluppatori, che fornisce un certificato temporaneo con il quale si può firmare un'applicazione. Grazie a questo servizio è possibile firmare tutte le applicazioni che si vogliono utilizzare. L'operazione deve essere ripetuta però, per tutti i telefoni sui quali si desidera installare il software e comunque non è il metodo "ufficiale". L'unica soluzione ufficiale è la firma dell'applicazione con metodi classici, cioè acquistando un certificato da enti autorizzati (CA) come ad esempio Verisign[16] o Thawte[17] a prezzi abbastanza eccessivi.

2.3. Java Micro Edition

Java Micro Edition fornisce un ambiente robusto e flessibile per le applicazioni per dispositivi mobili o altri telefoni, palmari e stampanti. Java ME include interfacce utenti flessibili, un sistema di sicurezza robusto e protocolli di rete incorporati. Le applicazioni basate su Java ME sono portabili su molti dispositivi che conservano le stesse caratteristiche hardware. La piattaforma J2ME è destinata a due grandi famiglie di prodotti con caratteristiche hardware differenti.

[16] http://www.verisign.com
[17] http://www.thawte.com

2.3.1. Configuration

Una *Configuration* è una descrizione completa di un Java Runtime per una determinata categoria di dispositivi con caratteristiche hardware simili. Essa è composta delle seguenti tre parti:

- Una JVM in grado di eseguire del *byte code*
- L'implementazione nativa di alcune interfacce per l'interazione con il SO su cui la JVM è in esecuzione
- Un insieme di classi core (API) standard

Al momento, J2ME definisce due *Configuration*:

- CDC: Connected Device Configuration prevede:
  - CPU a 32 bit
  - Almeno 512 KiloBytes di memoria ROM
  - Almeno 256 KiloBytes di memoria RAM
  - Alimentazione teoricamente illimitata
  - Connettività sempre disponibile
  - Disponibilità di un implementazione completa della JVM
  - Possibile presenza di un interfaccia grafica con l'utente

  A questa categoria appartengono dispositivi quali telefoni abilitati al Web, sistemi di navigazione, Web Tv e molti altri.
- CLDC Connected Limited Device Configuration prevede:
  - Una CPU più lenta (8-32 MHz) a 16 o 32 bit
  - Bassa disponibilità di alimentazione (a batterie)
  - Almeno 192 KiloBytes di memoria totale
  - Connettività limitata
  - Interfaccia grafica di qualsiasi tipo, anche assente

  A questa categoria appartengono telefoni cellulari, palmari e molti altri.

Nella figura sottostante si possono osservare le diverse configurazioni presenti attualmente.

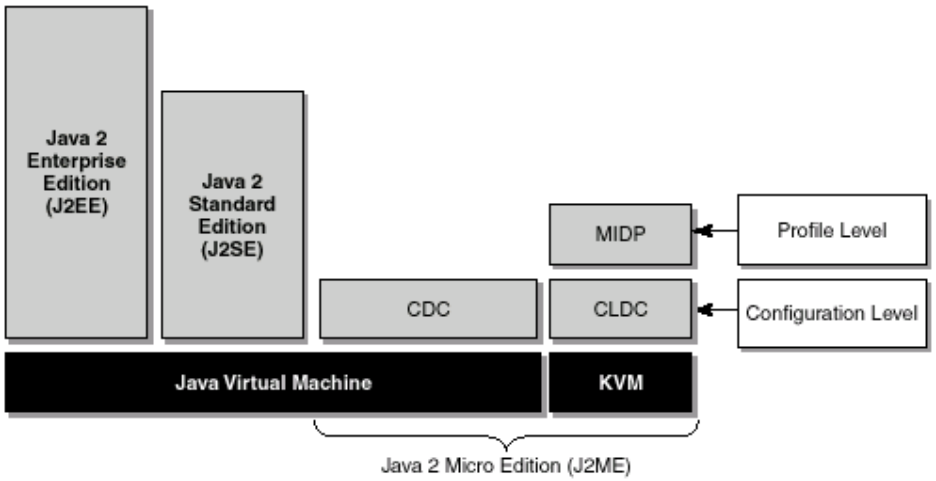


Figura 2.3.1.1: *Diversi Configuration presenti al momento.*

La relazione tra la J2SE[18], CDC e CLDC è espressa bene dalla figura 2.3.1.2. Si può notare come CDC sia un estensione della J2SE dal momento in cui implementa delle classi ad hoc per i vari dispositivi, e come CLDC sia un sottoinsieme in senso stretto di CDC.

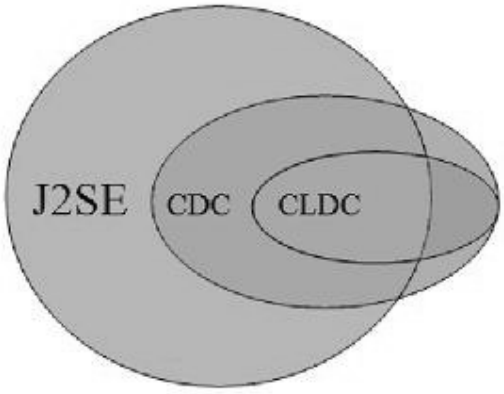


Figura 2.3.1.2: *Le relazioni tra le varie Configuration*

[18] http://java.sun.com/javase/index.jsp

2.3.2. Profile

L'insieme di API costruite a partire da una *Configuration* per un determinato tipo di dispositivi con lo scopo di sfruttarne tutte le caratteristiche hardware è detto *Profile.* Nella figura 2.3.2.1 è rappresentata l'architettura di un CLDC.

Figura 2.3.2.1: *Architettura di CLDC*

Tutte le applicazioni scritte per un determinato Profile dovranno quindi poter essere eseguite su tutti i dispositivi che supportino lo stesso Profile.

Il progetto è stato sviluppato utilizzando il Profile *Mobile Information Device Profile* (MIDP) versione 2.0 (JSR-118[19]) che è supportato da tutti i dispositivi aventi sistema operativo Symbian 8.0 o successivi.

2.3.3. La KVM

Le caratteristiche principali della CLDC sono definite nella *CLDC Specification 1.1* o anche detta CLDCS. Esse descrivono le caratteristiche fondamentali che deve possedere una Virtual Machine per poter eseguire applicazioni destinati a dispositivi con risorse

[19] http://www.jcp.org/en/jsr/detail?id=118

limitate. La Virtual Machine, attualmente disponibile, che soddisfa le specifiche CLDCS si chiama KVM (La K indica che si tratta di una Virtual Machine che ha bisogno di memoria dell'ordine dei KiloBytes). Si tratta di una Virtual Machine compatta e portabile, progettata per soddisfare i seguenti punti:

- Il suo codice non deve superare 80 KiloBytes.
- Deve essere perfettamente portabile.
- Deve essere progettata in modo modulare e scalabile.
- Deve essere più completa e più veloce possibile

2.3.4. J2SE e J2ME a confronto

Molte delle funzionalità della J2SE Virtual Machine sono state eliminate sia perché troppo onerose da implementare sia per motivi di sicurezza. Le limitazioni adottate nelle JVM per CLDC sono:

- Non esiste un supporto per i numeri a virgola mobile perché non è disponibile il supporto hardware.
- Non esiste la funzione *Object.finalization.*
- La cattura degli errori è limitata. Esistono solo tre tipi di errori.
- Non esiste la *Java Native Interface* (JNI[20])
- Non esiste il *calss-loader* definito dall'utente per motivi di sicurezza.
- Non è supportata la *reflection* e *serialization*.
- Non possono essere utilizzati *gruppi di threads* o *daemon threads*.
- Non esiste *weak-reference*.

Alcune classi sono state ereditate dalla J2SE ed alcune sono state implementate appositamente per sfruttare le caratteristiche specifiche dei telefonini. Nelle seguenti tabelle sono elencate le due categorie.
Classi ereditate:

[20] http://java.sun.com/j2se/1.4.2/docs/guide/jni/

Package	Classi
java.lang	Boolean, Byte, Character, Class, Integer, Long, Math, Object, Runnable, Runtime, Short, String, StringBuffer, System, Thread, Throwable
java.io	ByteArrayInputStream, ByteArrayOutputStream, DataInput, DataOutput, DataInputStream, DataOutputStream, InputStream, OutputStream, InputStreamReader, OutputStreamWriter, PrintStream, Reader, Writer
java.util	Calendar, Date, Enumeration, Hashtable, Random, Stack, TimeZone, Vector

Classi specifiche per CLDC:

Package	Classi
javax.microedition.io	Connection, ConnectionNotFoundException, Connector, ContentConnector, Datagram, DatagramConnection, InputConnection, OutputConnection, StreamConnection, StreamConnectionNotifier

2.3.5. Sicurezza

La sicurezza nell'ambiente J2ME si basa sulla tecnica già spiegata nel capitolo 2.2.1. ed in particolare sulla firma digitale delle applicazioni. E' un modello che viene anche chiamato *sandbox,* perché viene concessa l'esecuzione di ogni programma ma con privilegi differenti a seconda delle credenziali che essi possiedono. Dal

punto di vista della gestione e dell'amministrazione la sandbox è composta da seguenti elementi principali:

- Permessi: sono uno strumento per regolamentare e proteggere l'accesso a particolari API che richiedono la specifica conferma di accesso ad esse. Un esempio potrebbe essere l'accesso ai dati utilizzando l'API *javax.microedition.io.Connector.open().*
- domini di protezione: essi definiscono un insieme di permessi e sono il concetto di base della sandbox. Ad ogni tipo di applicazione viene associato un dominio, cioè un insieme di API che può invocare.
- domini di policy: consiste nel definire un *alias* per un insieme di permessi logicamente correlati (ad es. tutti i permessi relativi all'accesso ai dati utente).
- Keystore: è una locazione di memoria dove vengono memorizzati i certificati digitali utilizzati per la firma delle applicazioni. A seconda se l'applicazione utilizzata possiede un certificato valido verrà assegnato un dominio di protezione differente.

Le *MIDlet*, cioè programmi scritti in J2ME, possono essere di tipo *trusted* o *untrusted*. Un'applicazione è di tipo trusted se la sua firma digitale è presente nella Keystore e le viene assegnato il dominio di esecuzione descritto nella tabella 2.3.4.1. Un'applicazione è di tipo untrusted se non è certificata o se il suo certificato non è valido e il suo dominio è descritto nella tabella 2.3.4.2. Le celle che contrassegnate da "default" rappresentano lo stato al momento del'installazione. Gli altri stati possono essere impostati se le celle contengono "si".

	Accesso negato	**Chiedi sempre**	**Chiedi prima volta**	**Sempre consentito**
Accesso alla rete	si	si	default	si
Messaging	si	default	-	-
Autostart delle applicaz.	si	si	default	si
Connettività	si	-	default	si
Multimedia	si	-	default	si
Lettura dati utente	si	default	si	si
Modifica dati utente	si	default	si	si
Attivazione chiamate	si	default	-	-

Tabella 2.3.4.1: A*pplicazioni di terze parti firmate (trusted).*

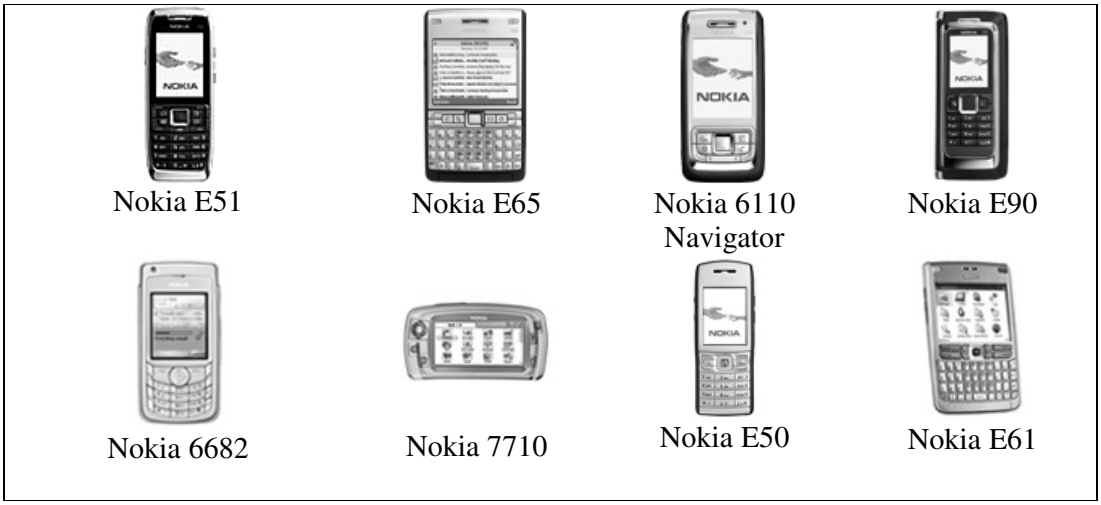
	Accesso negato	**Chiedi sempre**	**Chiedi prima volta**	**Sempre consentito**
Accesso alla rete	si	default	si	-
Messaging	si	default	-	-
Autostart delle applicaz.	si	si	default	-
Connettività	si	-	default	si
Multimedia	si	default	si	-

Lettura dati utente	default	-	-	-
Modifica dati utente	default	-	-	-
Attivazione chiamate	si	default	-	-

Tabella 2.3.4.2: A*pplicazioni di terze parti non firmate (untrusted).*

2.4. Panoramica sui vari modelli disponibili

Attualmente in commercio esistono molti modelli di telefonini con sistema operativo Symbian. I modelli elencati nella tabella sottostante hanno il sistema operativo Symbian OS S60 e appartengono alla famiglia di prodotti che hanno tutte le caratteristiche necessarie all'esecuzione del applicazione sviluppata. Per affermare con sicurezza che l'applicazione funzioni correttamente si dovrebbe effettuare il *testing* su ogni modello perché in tanti casi si sono verificati problemi specifici per vari modelli con qualche libreria Java o di altro tipo. Nel capitolo successivo verranno spiegati i motivi per cui l'applicazione sviluppata potrebbe non funzionare.




Nokia N70


Nokia N72


Nokia N73


Nokia E70


Nokia N76


Nokia N80


Nokia N81


Nokia N82


Nokia N90


Nokia N91


Nokia N92


Nokia N93


Nokia N93i


Nokia N95


Nokia N95 8GB


LG Joy


Panasonic X800


Motorola A1000


Motorola MOTORIZR Z8


Samsung SGH-i450


Samsung SGH-i550


Samsung SGH-i560


Samsung SGH-D720


Samsung SGH-D730


Sony Ericsson M600i


Sony Ericsson P1i


Sony Ericsson P990


Sony Ericsson W960

2.5. Requisiti necessari al funzionamento

Il progetto è stato sviluppato secondo le specifiche CLDC 1.1 e utilizzando le API della MIDP 2.0. Dovrebbe quindi essere compatibile con ogni dispositivo che abbia queste caratteristiche ma non è così. I motivi per cui ciò non è vero sono molteplici:

- disponibilità di memoria fisica: non tutti i telefonini possiedono abbastanza memoria per contenere le presentazioni e in molti non può essere espansa con le memory card
- disponibilità di memoria RAM: molti telefoni non hanno un sufficiente quantitativo di memoria RAM. Le prime versioni dell'applicazione erano molto più "sensibili" su questo fatto ma a causa dei motivi che sono spiegati nel punto successivo si è scelta un'implementazione più semplice, ma che necessita di molta memoria RAM.
- errori di funzionamento della KVM: alcune API di controllo dell'audio e video non sono state implementate correttamente nelle prime versioni. Ad esempio `Player.setMediaTime()` è un istruzione che non è stata implementata correttamente in molti dispositivi (compreso il primo utilizzato nello sviluppo). Pertanto non è possibile utilizzare l'applicazione in essi.

3. Requisiti e Funzionalità implementate

3.1. Requisiti funzionali

Lo scopo del progetto è di emulare le caratteristiche principali del sistema LODE. Deve essere adattato all'ambiente di sviluppo utilizzato e deve tener conto delle caratteristiche hardware dei dispositivi su cui verrà eseguito. Le principali funzionalità offerte dal sistema LODE sono:

- visualizzazione del video della presentazione
- visualizzazione delle presentazioni in PowerPoint sincronizzate con il video.
- Possibilità di ingrandire la dimensione delle immagini.
- Possibilità di mettere in pausa e ripristinare una presentazione.
- Possibilità di navigazione attraverso i vari capitoli in cui è suddivisa una presentazione.
- Possibilità di posizionamento in un determinato punto della presentazione.
- Controllo del volume

Questi sono quindi i requisiti funzionali del progetto. Nel capitolo 3.3 vedremo quali sono stati implementati e quali no.

3.2. Requisiti non funzionali

Anche i requisiti non funzionali sono uguali a quelli del sistema LODE ma hanno ciascuno un importanza diversa a causa delle caratteristiche hardware dei dispositivi sui quali verrà eseguita l'applicazione. Essi sono:

- Semplicità di utilizzo.
- Robustezza.

- Facilità di ottenimento delle presentazioni.
- Qualità delle presentazioni.
- Compatibilità con altri dispositivi.
- Velocità di caricamento delle presentazioni.

Gli ultimi 3 requisiti sono importanti per il sistema LODE, ma in questo progetto hanno un importanza cruciale. Essi sono correlati tra di loro. Ad esempio: aumentando la qualità delle presentazioni (qualità audio e risoluzione delle immagini) aumenta anche il tempo di caricamento, ed è probabile che venga escluso qualche dispositivo che non ha potenzialità hardware per gestire una presentazione di qualità impostata. Nella valutazione della relazione qualità – velocità - compatibilità si è tenuto conto della attuale diffusione dei telefonini e delle loro caratteristiche. La stessa valutazione non sarà più valida fra qualche anno e si dovranno rivedere le scelte fatte per ottenere un migliore rapporto.

3.3. Funzionalità implementate

Come già detto nel capitolo 3.1, le funzionalità dell'applicazione prodotta cercano di emulare le principali funzionalità dell'applicazione del sistema LODE. Nella figura 3.3.1 è rappresentata un immagine del funzionamento del sistema LODE con le caratteristiche principali.

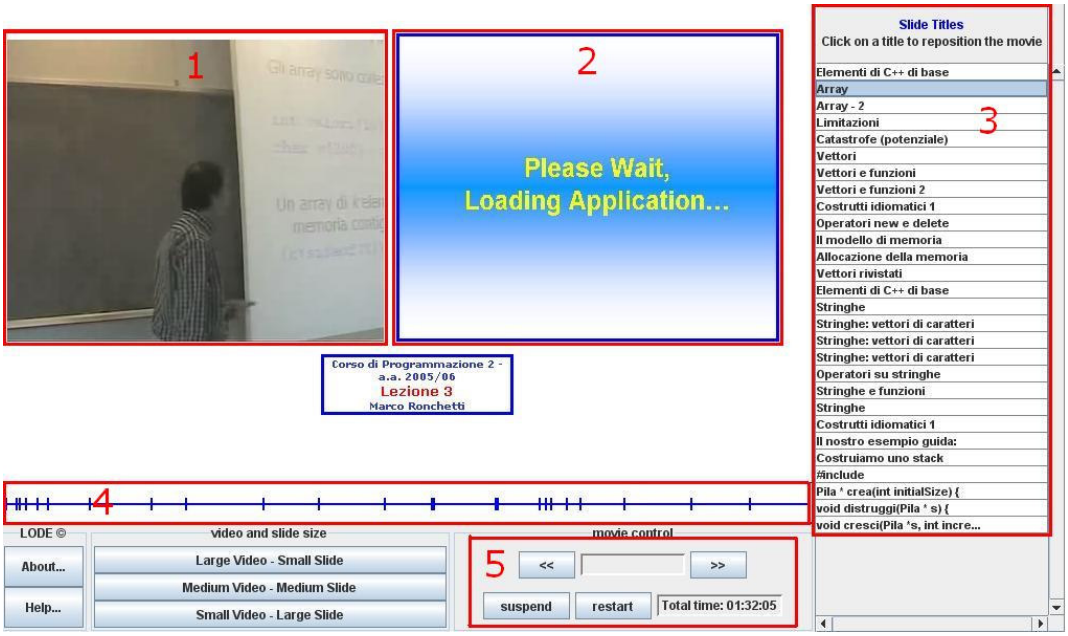


Figura 3.3.1: *L'interfaccia del sistema LODE.*

Leggenda:

1: Il video della lezione.

2: La presentazione PowerPoint associata.

3: I vari capitoli della presentazione.

4: Traccia audio navigabile.

5: Pulsanti di sospensione e ripristino della presentazione.

Le funzionalità di supporto implementate nel progetto sono:

- Riproduzione dell'audio delle presentazione.
- Visione delle presentazioni in PowerPoint mostrate durante la presentazione.
- Selezione dei capitoli in cui è suddivisa un'intera presentazione.
- Riavvolgimento veloce delle tracce audio in avanti e indietro.
- Controllo del volume.
- Aggiornamento della lista delle lezioni presenti nella memoria.

L'unica funzione che non è stato possibile aggiungere è la riproduzione del video. Esso richiedeva troppa memoria e i tempi di caricamento si allungavano notevolmente rendendo l'uso

impraticabile. Il video comunque non è strettamente necessario per il tipo di presentazioni che si intendono riprodurre in quanto in esse non viene mostrato nient'altro oltre alla presentazione PowerPoint. Pertanto sono sufficienti l'audio del parlato e le diapositive.

4. Automazione e conversione

Il processo di automazione della conversione consente di trasformare una lezione compatibile con sistema LODE in una compatibile con il player sviluppato nel progetto. E' un processo che viene eseguito in automatico da una serie di applicazioni, alcune sviluppate in ambiente J2SE e altre di terze parti. In seguito saranno elencati i vari programmi utilizzati e illustrati i principali passaggi.

4.1. Struttura di una presentazione

Le presentazioni hanno una loro struttura specifica adatta alla piattaforma di utilizzo. Ad esempio in quella utilizzata da LODE, le informazioni riguardanti i capitoli e i tempi di passaggio da una slide ad un'altra sono contenuti in un file html che verrà poi letto dal browser ed elaborato da degli script. Nella versione per telefonini tali informazioni devono essere contenute in un altro tipo di file e possibilmente insieme ad altre informazione per velocizzare i tempi di accesso e lettura. Nei capitoli successivi verranno spiegato in dettaglio come sono strutturati i due tipi di presentazioni.

4.1.1. Struttura di partenza

La struttura delle lezioni compatibili con LODE è illustrata nella Figura 4.1.1.1. I componenti principali sono:

- La cartella principale contenente un file *leggimi.html*, una cartella *common* e un'altra chiamata *content*. In quest'ultima sono presenti tutti i dati necessari per costruire una presentazione compatibile con il player sviluppato.

- Nella cartella *content* è presente un file *index.html* che contiene informazioni riguardanti i tempi e i modi in cui avviene il passaggio da un capitolo ad un altro.
- All'interno della cartella *data* è presente il video della lezione ripreso con una videocamera chiamato *movie.mp4*.
- Nella cartella *content_frame* sono contenute le immagini della presentazione PowerPoint in formato jpg[21].

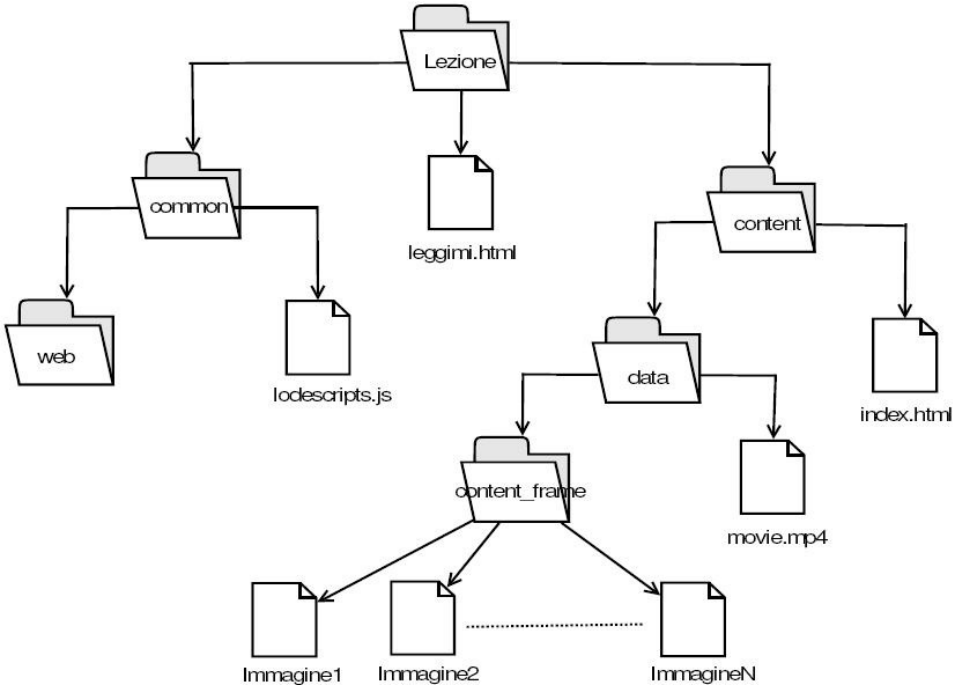


Figura 4.1.1.1: *Struttura della lezione di partenza*

4.1.2. Struttura compatibile con il Player

I files e le cartelle elencate in precedenza saranno utilizzati dai vari programmi per costruire una presentazione compatibile con il player. La sua struttura è descritta in seguito:

- Le tracce audio sono estratte dal file *movie.mp4,* dapprima in formato mp3 e successivamente suddivise in parti più corte e

[21] http://www.jpeg.org/

trasformate in formato *amr*[22]. Nel paragrado 4.2.1 è spiegato il motivo della scelta del formato amr.

Quando una traccia viene riprodotta in ambiente JavaME, essa viene prima caricata completamente in memoria. Non è di conseguenza possibile riprodurre tracce che superano la dimensione della memoria RAM complessiva del dispositivo. E' stato necessario quindi suddividere l'audio della presentazione in parti di dimensioni più piccole. Il metodo di suddivisione verrà spiegato nel Capitolo 4.4.

- Un file *dati.lez* che contiene tutte le informazioni sui capitoli estratte dal file *index.html* e le immagini ridimensionate della presentazione PowerPoint. La sua struttura è illustrata nella figura 4.1.2.2.

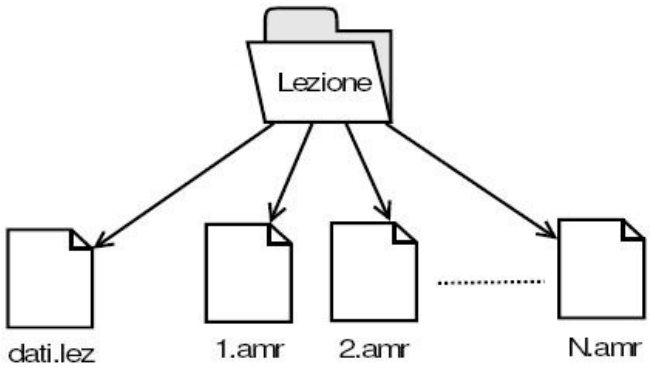


Figura 4.1.2.1: *Struttura compatibile con il player sviluppato*

[22] http://en.wikipedia.org/wiki/Adaptive_multi-rate_compression

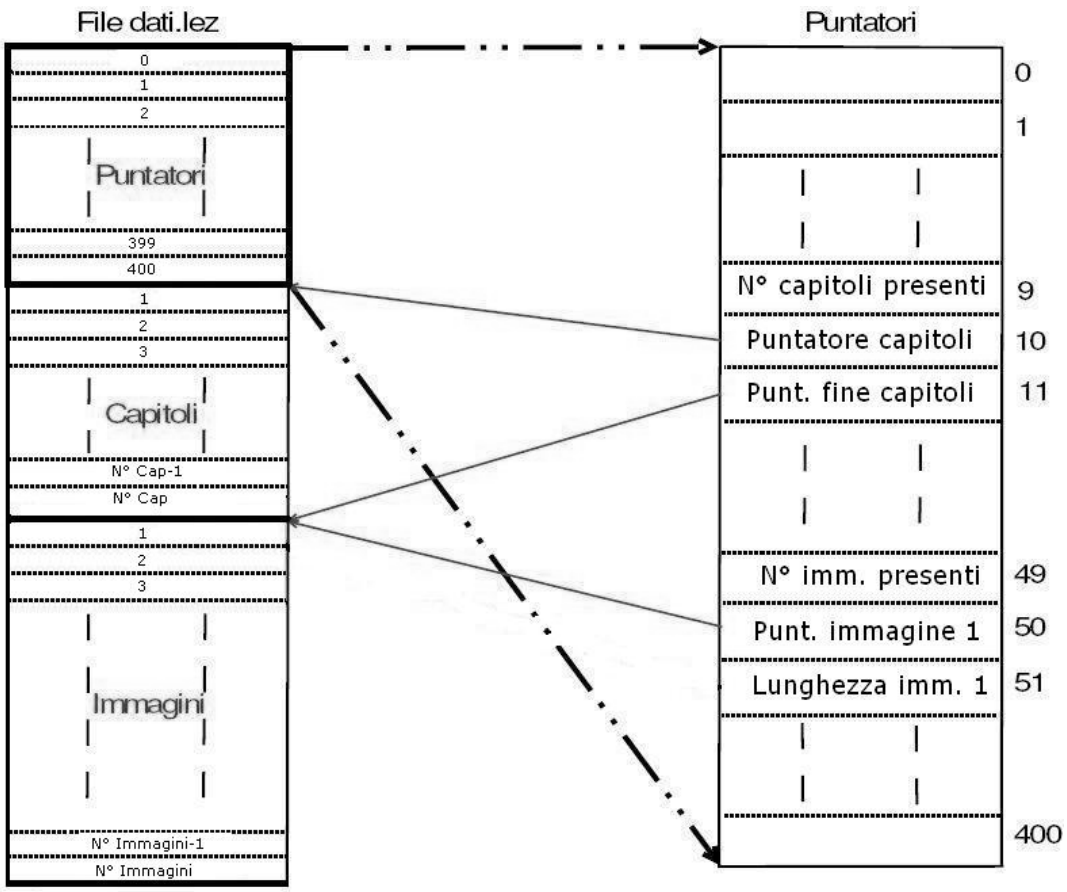


Figura 4.1.2.2: *Struttura file dati.lez e dei puntatori*

La decisione più importante riguardo alla logica della parte di automazione è stata sicuramente la scelta della struttura con cui rappresentare il file *dati.lez*. Innanzitutto è stato necessario unire in un file tutte le informazioni riguardanti i capitoli e le immagini a causa dei problemi con i permessi relativi all'applicazione sviluppata. In caso in cui si fosse utilizzata una struttura uguale a quella di partenza (un file per i capitoli ed ogni immagine separata) si sarebbe dovuto dare conferma di accesso ai dati utente ad ogni lettura di files. Una soluzione simile non è accettabile. La soluzione proposta prevede di unire le immagini e i capitoli in un unico file limitando il numero di richieste di permessi ad una sola. Per fare ciò è stato creato un archivio personalizzato che dà la possibilità di accesso diretto per la

lettura di informazioni necessarie. Per accedere ai dati è stato necessario creare un indice che fornisca informazioni sui dati in esso contenuti. La soluzione proposta utilizza un metodo semplice ed intuitivo e cioè un array di puntatori che tiene traccia sulla posizione in cui sono immagazzinati i capitoli e le immagini ed inoltre contiene altre informazioni aggiuntive come ad esempio il numero delle immagini e capitoli presenti. Nella figura 4.1.2.2 è illustrata la struttura utilizzata.

Nel file *dati.lez* sono contenute anche le informazioni riguardanti i capitoli. Esse vengono estratte dal file *index.html* e vengono scritte in sequenza dopo i puntatori. La struttura dei capitoli è rappresentata in figura 4.1.2.3.

Figura 4.1.2.3: *Struttura di un capitolo.*

4.2. Programmi & Codec utilizzati

Per effettuare la conversione in automatico sono stati utilizzati vari programmi di terze parti, preferibilmente gratuiti, ed alcuni implementati ad-hoc. E' stato necessario decidere anche i formati in cui rappresentare i dati. Queste scelte hanno avuto un

importanza cruciale perché, oltre ad incidere sulle prestazioni del player sviluppato, definiscono implicitamente anche la compatibilità con i vari tipi di dispositivi. Nei prossimi capitoli sono spiegate le varie scelte.

4.2.1. Programmi

I programmi utilizzati per le conversioni ed elaborazioni sono:

- *GestioneArchivio.jar*: è il programma scritto il Java che permette di ottenere informazioni sulla divisione in capitoli dell'intera lezione. Crea inoltre un archivio contenente tutte le immagini e gli indici dei capitoli che verranno poi utilizzati dal player. I passaggi principali verranno spiegati nei capitoli successivi.
- *IrfanView*[23]: è un software di elaborazione di immagini con licenza freeware[24] che si può utilizzare anche da linea di commando.
- *mp3splt*[25]: è un programma molto semplice che permette di suddividere una traccia audio in più parti senza effettuare la decodifica. Quindi è molto veloce e leggero.
- *ffmpeg*[26]: è un programma di elaborazione audio e video con licenza LGPL[27] che permette di estrarre e modificare l'audio da una traccia video.
- *Conversione.bat*: tutti i programmi elencati sopra effettuano una sequenza di operazioni nella conversione di una lezione. Tale sequenza è specificata nel file conversione.bat.

[23] http://www.irfanview.com/
[24] http://it.wikipedia.org/wiki/Freeware
[25] http://mp3splt.sourceforge.net/mp3splt_page/home.php
[26] http://ffmpeg.mplayerhq.hu/
[27] http://www.gnu.org/copyleft/lesser.html

4.2.1. Codecs

Il video delle lezioni di partenza è in formato mp4. Con l'ausilio di ffmpeg avviene l'estrazione dell'audio dal video e la conversione in un formato più "leggero" per poterlo leggere in ambiente JavaME. I codecs supportati dalla maggior parte dei telefonini aventi sistema operativo Symbian OS sono rappresentati nella tabella 4.2.1.1.

	AMR	**AMR WB**	**Real Audio**	**MP3**	**AAC-LC, AAC-LTP**	**eAAC+, AAC+**	**WAVE**
	.amr	.awb	.ra, .rm	.mp3	.aac, .mp4, .3gp, .m4a	.3gp, .mp4	.wav
Series 40 2st edition Nokia 3220,5140i, 6030, 6101	SI Bitrate (4.75 – 12.2)	SI Bitrate (6.60 – 23.85)	NO	NO	.aac	NO	NO
Series 40 3nd edition 6111,6280, 7370	SI Bitrate (4.75 – 12.2)	NO	NO	SI	.aac e alcuni .m4a	NO	SI
S60 1st edition 3650, N-Gage QD	SI Bitrate (4.75 – 12.2)	SI Bitrate (6.60 – 23.85)	NO	NO	NO	NO	SI
S60 2nd edition	SI Bitrate	SI Bitrate	SI	SI	.aac, .3gp,	SI	SI

6680, N70, N90	(4.75 – 12.2)	(6.60 – 23.85)			.mp4		
S60 3rd edition E60,E61	SI Bitrate (4.75 – 12.2)	SI Bitrate (6.60 – 23.85)	SI	SI	.aac, .3gp, .mp4	SI	SI
S80 2nd edition 9300, 9500	SI Bitrate (4.75 – 12.2)	SI Bitrate (6.60 – 23.85)	SI	SI	.aac, .3gp, .mp4	SI	SI

Tabella 4.2.1.1: *Modelli e codecs supportati* [NF1]

Come si può notare solo il formato AMR (Adaptive Multi-Rate) è supportato da quasi tutti i telefonini con sistema operativo Symbian OS. E' un codec adatto alla compressione di tracce audio contenenti il parlato e grazie ad esso è possibile ridurre le dimensioni delle tracce audio di circa dieci volte rispetto ad una traccia mp3. Soddisfa, quindi, tutte le caratteristiche necessarie al progetto. Il software utilizzato per la conversione è ffmpeg.

Per quanto riguarda le immagini invece, la loro dimensione di partenza è 1024 x 768 pixel in formato JPEG ed occupano circa 100 KiloBytes ciascuna. La loro dimensione è eccessiva per essere visualizzate sui telefonini che hanno spesso lo schermo di 176 x 208 pixel. Inoltre riducendo la loro grandezza è possibile accelerare il tempo di caricamento. Ricordiamo che dimezzando l'altezza e la larghezza di un immagine, lo spazio occupato diminuisce di quattro volte e quindi anche il tempo di caricamento. Utilizzando il formato JPEG questo rapporto non è necessariamente esatto ma dipende dalla qualità di approssimazione che si desidera. Riducendo la dimensione si ottiene comunque un incremento di prestazioni ma una perdita dei dettagli delle immagini. Effettuando delle prove si è deciso che utilizzando immagini di dimensioni doppie rispetto allo schermo si riuscivano a leggere tutti i dettagli. La codifica utilizzata prevede

immagini di dimensioni 312 x 416 pixel in formato JPEG con una qualità di compressione del 75%. Si ottengono così immagini che occupano dai 10 ai 20 KiloBytes. Il processo è completamente automatico e viene effettuato utilizzando IrfanView.

4.3. Passaggi principali

La procedura di conversione è gestita dalla sequenza di comandi contenuta nel file *Conversione.bat*. La figura 4.3.1 rappresenta lo schema dei passaggi principali.

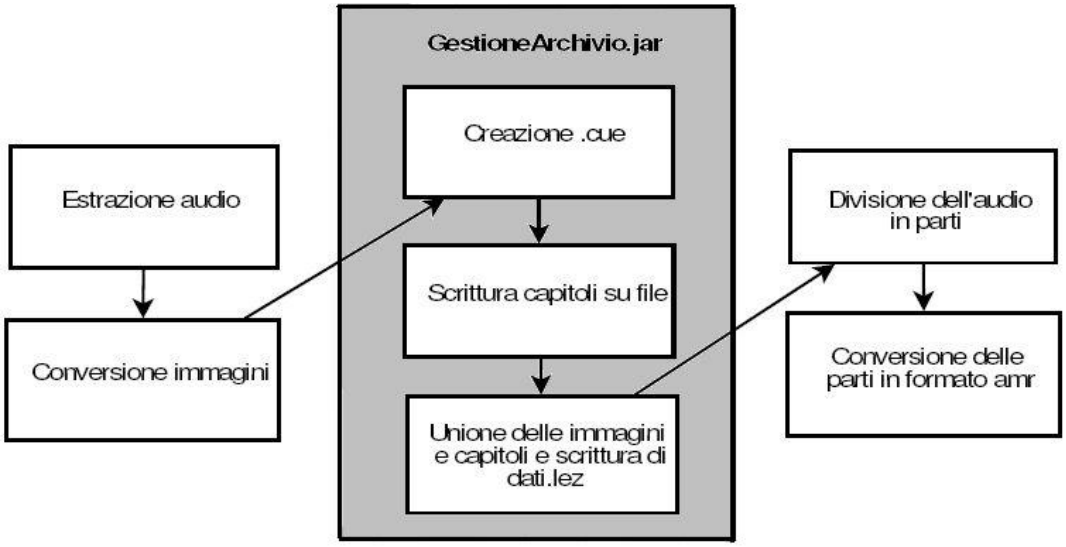


Figura 4.3.1: *Passaggi principali nella conversione delle presentazioni*

La sequenza delle operazioni:

- Ridimensionamento e rotazione delle immagini contenute nella cartella *content_frame* della lezione di partenza:

  Sintassi di utilizzo di IrfanView:

  `i_view32 [sorgente] [options]…[options] [destinazione]`

  Commando utilizzato:

.\Programmi\IrfanView\i_view32 \content_frame*.jpg /rotate_l /resize=(312,416) /resample /aspectratio /jpgg=75 /convert=.\Images*.jpg

\content_frame*.jpg : Immagini di partenza.
/rotate_l : Ruota le immagini verso sinistra perché la visione sui telefonini sarà fatta orizzontalmente.
/resize=(312,416) : Riduce le dimensioni delle immagini a 312 x 416 pixels.
/resample : Migliora la qualità delle immagini ottenute in caso di ridimensionamento.
/aspectratio : Mantiene le proporzioni delle immagini migliorandone la qualità.
/jpgg=75 : Imposta la qualità con la quale verranno salvate le immagini.
/convert=.\Images*.jpg : cartella e estensione con la quale verranno salvate.

- Estrazione dell'audio dal file movie.mp4 tramite ffmpeg:

Sintassi di utilizzo di ffmpeg:
ffmpeg [[opzioni file sorgente] –i file sorgente]... {[opzioni file destinazione] file destinazione}

Commando utilizzato:
.\Programmi\ffmpeg\ffmpeg.exe –i movie.mp4 –vn –ab 128 movie.mp3

–i movie.mp4 : Specifica il file in ingresso.
–vn : Esclude il video dalla conversione.
–ab 128 : Specifica il bitrate del file risultante.

movie.mp3 : Il nome e il tipo del file in uscita.

- Esecuzione del programma GestioneArhivio.jar che compie le seguenti azioni:
  - Lettura del file index.html e creazione di una compilation *compilation.cue* che verrà utilizzata successivamente dal programma mp3split per suddividere la traccia audio intera i parti più piccole.
  - Lettura delle immagini convertite e la seguente scrittura nel file dati.lez sotto forma di arrays di bytes contigui.
  - Scrittura dei puntatori nel file dati.lez.

  I passaggi verranno descritti in dettaglio nel capitolo 5.

- Suddivisione della traccia intera in parti più piccole con il programma mp3split.

  Sintassi di utilizzo di mp3splt:

  mp3splt [opzioni] file [tempo_di_inizio] [temp_di_fine]

  Comando utilizzato:

  .\Programmi\mp3splt-2.1\mp3splt.exe -f -c compilation.cue -d Audios movie.mp3 -n -o @n@t

  -f : viene fatta la divisione processando i frames. Si ottiene in questo modo una precisione migliore.

  -c compilation.cue : imposta compilation.cue come la compilation secondo la quale verrà effettuata la divisione.

  Le compilation sono dei semplici file di testo ed hanno la seguente forma:

  TRACK n type
  INDEX m time

n : Numero della traccia

type : Tipo di file (AUDIO nel nostro caso)

m : Numero del pezzo ottenuto.

time : Tempo di separazione.

-d Audios : imposta la cartella di destinazione dei files ottenuti.

movie.mp3 : file da elaborare.

-n : non imposta nessun tipo di *Tag* nel file ottenuti.

-o @n@t : specifica il formato del nome dei file ottenuti. **@n** indica il numero della traccia mentre **@t** il titolo.

- Conversione delle tracce ottenute in formato *amr* tramite ffmpeg:

  Comando utilizzato:

  ```
  for %%v in (Audios\*.mp3) do .\programmi\ffmpeg\ffmpeg.exe -i %%v -ac 1 -ar 8000 -ab 12.2k -f amr -acodec libamr_nb %%v.amr
  ```

  for %%v in (Audios*.mp3) do : ciclo con il quale il commando viene applicato ad ogni file con estensione .mp3 nella cartella Audios.

  -i %%v : specifica i file in ingresso.

  -ac 1 : impost il numero di canali che avrà il file in uscita.

  -ar 8000 : imposta la frequenza di campionamento del file in uscita.

  -ab 12.2k : imposta il bitrate del flusso audio in uscita.

  -f amr : imposta il tipo del file prodotto.

  -acodec libamr_nb : specifica quale tipo di codec utilizzare.

%%v.amr : specifica il nome con il quale verranno salvati i files ottenuti.

4.4. Considerazioni generali

Tutti i passaggi vengono completati in pochi minuti. La parte più onerosa è l'estrazione dell'audio dal video. Per la restante parte ci vogliono pochi secondi di attesa. La qualità ottenuta dipende da molti parametri, partendo dalla qualità audio impostata fino alla dimensione delle immagini. Tutte le impostazioni tengono conto dei limiti hardware e software dei telefonini, quindi in un futuro breve tali parametri potranno essere modificati per sfruttare meglio le caratteristiche dei nuovi telefonini.

C'è da considerare anche la questione dei permessi delle applicazioni. Se avessimo tutti i permessi si avrebbe una qualità sia audio che di immagini migliore e si potrebbero saltare alcuni passaggi della conversione come ad esempio la costruzione dei capitoli e dei puntatori. Si potrebbero utilizzare direttamente i files delle presentazioni di partenza.

5. Player

Il player è l'applicativo sviluppato in J2ME che riproduce le presentazioni sui telefonini. Nei seguenti capitoli verrà illustrata la sua struttura e i programmi utilizzati nello sviluppo. Con il termine player sarà indicato il programma sviluppato, mentre il Player è l'interfaccia Java che permette di riprodurre l'audio.

5.1. Programmi utilizzati

Per lo sviluppo dell'applicazione è stato utilizzato NetBeans IDE 5.5.1 con l'integrazione di un insieme di strumenti per lo sviluppo di applicazioni in J2ME chiamato WTK 2.5.1[28] (Sun Wireless Toolkit 2.5.1). Esso offre sia le librerie del Profile MIDP 2.0 sia strumenti per il debugging, misurazione dell'efficienza ed emulazione dei dispositivi mobili. A causa della mancanza di alcune impostazioni personalizzate per i telefonini Nokia è stato utilizzato un altro strumento per l'emulazione del player. Si chiama Carbide.j[29] ed offre anch'esso sia le librerie necessarie per lo sviluppo sia uno strumento per l'emulazione. E' possibile installarlo sia in modalità indipendente sia integrandolo con NetBeans. Può essere utilizzato inoltre per firmare le applicazioni prodotte se si è in possesso di un certificato valido. Nel caso del Symbian OS 8.0 è possibile produrre un proprio certificato, installarlo nel telefonino e firmare le applicazioni. Nelle versioni successive non è possibile installare i propri certificati, quindi questo metodo non può essere utilizzato.

[28] http://java.sun.com/products/sjwtoolkit/download-2_5_1.html
[29] http://www.forum.nokia.com/main/resources/tools_and_sdks/carbide/index.html

5.2. Struttura Player

Il player, come già detto nel capitolo 3.3, riproduce l'audio e le immagini della presentazione offrendo delle funzionalità di supporto come scelta dei capitoli o riavvolgimento dell'audio ecc. Queste funzionalità sono offerte da speciali API del Profile MIDP 2.0 che si chiamano Mobile Media API (JSR 135[30]). Esse offrono una gran quantità di funzioni per manipolare tracce audio e video e sono suddivise in due *package:*

- javax.micromedia.media
- javax.micromedia.media.control

La struttura dei due package è illustrata nella sottostante figura.

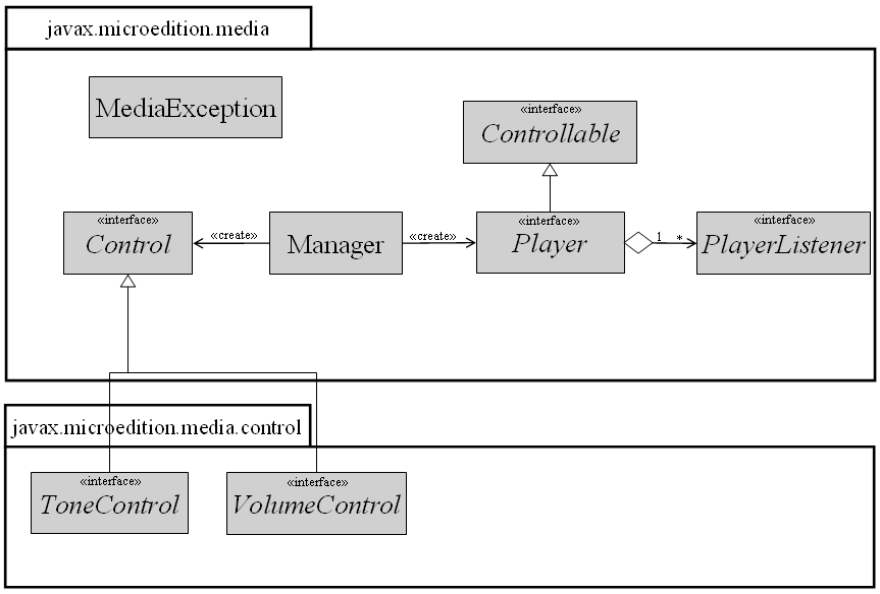


Figura 5.2.1: *Struttura dei package di MMAPI*

Gli elementi fondamentali dei due package e dell'applicazione sono i seguenti tre elementi che interagiscono tra di loro:

- Manager
- Player

[30] http://java.sun.com/products/mmapi/overview.html

- Control

La Tabella 5.2.2 raffigura il modo in cui interagiscono tra di loro ma in seguito verranno spiegate in dettaglio le loro caratteristiche e funzionalità offerte.

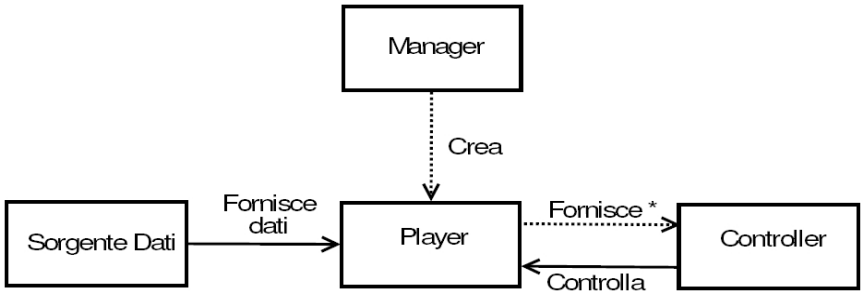


Tabella 5.2.2: *Diagramma di iterazione tra i componenti.*

5.2.1. Manager

Il Manager rappresenta il punto d'ingresso per l'utilizzo delle MMAPI ed è la classe principale che consente di:

- Interrogare il dispositivi sui protocolli e sulle tipologie di contenuti multimediali supportati.
- Ottenere il riferimento alla realizzazione dell'interfaccia Player in grado di riprodurre il media.
- Riprodurre un singolo tono di frequenza, durata e volume specificati.

La funzionalità più importante è sicuramente la possibilità di creare un istanza del Player. Il Player può essere invocato su diversi tipi di media. Esiste perciò un metodo standard per specificare che tipo di media si vuole riprodurre chiamato *locator.* Esso ha la seguente sintassi:

`[protocol]:[content location]` dove *protocol* indica il protocollo da utilizzare e *content location* indica l'indirizzo al quale si trova la risorsa.

Esempio:

http://science.unitn.it/ciao.wav

Per creare il Player si può utilizzare il seguente comando:

```
Manager.createPlayer(locator);
```

Un altro tipo di sintassi può essere utilizzato in caso in cui la risorsa si trovi sul *File System* o in un archivio. Essa prevede l'ottenimento dell'*InputStream* dalla risorsa e la specifica del *contentType*. Il *contentType* è una stringa che identifica il tipo del file da leggere. Nel nostro caso è stata utilizzata la seguente sequenza di comandi:

```
1 try {
2 String ctype = "audio/amr";
3 String url = "E:/Lezione1/1.amr";
4 FileConnection fc = (FileConnection) Connector.open(
5 url,Connector.READ);
6 InputStream in = fc.openInputStream();
7 player = Manager.createPlayer(in, ctype);
8 } catch (Exception ex) { }
```

In questo caso il *contentType* viene specificato nella riga 2 e l'*InputStream* è ottenuto nella riga 6. Il Player viene creato nella riga 7 mentre eventuali errori vengono catturati nella riga 8.

5.2.2. Player

Nel capitolo precedente sono stati illustrati i vari metodi per creare un Player. In seguito verranno illustrate le sue caratteristiche e le funzionalità.

Un Player per riprodurre un suono ha bisogno di acquisire delle risorse e talvolta questa operazione è molto onerosa. E' molto importante quindi che l'applicazione conosca in ogni momento lo stato di un Player in modo da poter eseguire operazioni dispendiose in momenti più opportuni. A tale scopo sono stati definiti alcuni stati e dei metodi che permettono di passare da uno stato ad un altro.

I vari stati sono:

- Unrealized: E' lo stato iniziale. Il Player non conosce le risorse di cui necessita per la riproduzione del media.
- Realized: Si trova in questo stato quando è a conoscenza di tutte le risorse necessarie.
- Prefetched: Si trova in questo stato quando è pronto per l'esecuzione.
- Started: In questo stato sta riproducendo il media.
- Closed: Ha concluso la riproduzione e ha liberato la memoria. Non può essere più riutilizzato.

Nella Figura 5.2.2.1 sono rappresentati tutti gli stati e i metodi che è necessario invocare per raggiungerli.

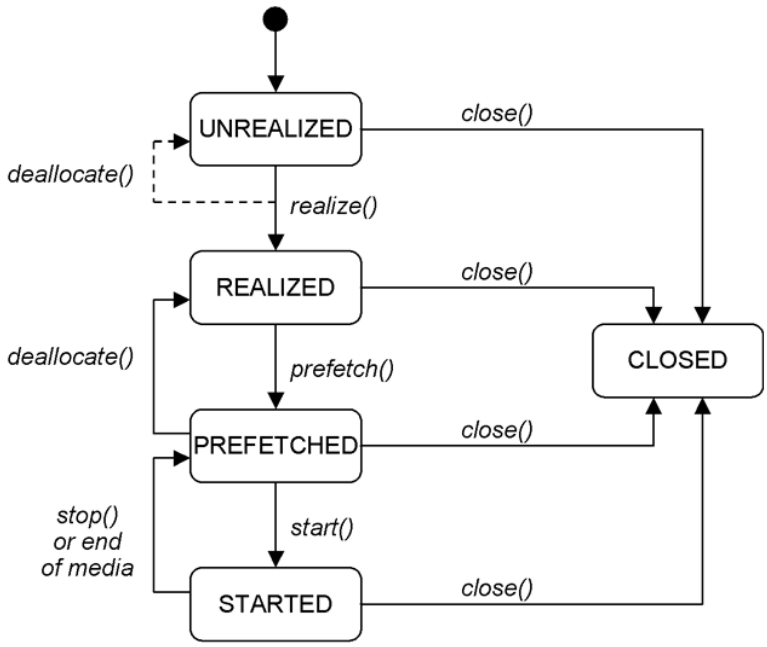


Figura 5.2.2.1: *Stati e metodi di un Player*

Tramite l'interfaccia *PlayerListener* è possibile monitorare gli stati di un Player.
Esistono inoltre metodi che permettono di impostare dei parametri al Player come ad esempio il metodo *setMediaTime(time)* che permette di posizionarsi all'interno della traccia in un determinato istante specificato dal parametro *time.* Come già accennato in precedenza durante lo sviluppo del progetto è stato scoperto un errore di implementazione del metodo in alcuni modelli.

5.2.3. Controller

Il Player fornisce due interfacce di controllo che sono *VolumeControl* e *ToneControl* ed entrambe fanno parte del package *javax.micromedia.media.control.* E' possibile ottenere il riferimento al *Control* desiderato invocando il metodo *Player.getControl(ConotrolType)* dove *ControlType* indica il tipo di controllo.
Esempio:

```
VolumeControl vc = (VolumeControl)
Player.getControl("VolumeControl");
vc.setLevel(50);
```

5.2.4. Il Design Pattern MVC

In informatica il Design Pattern è definito come una soluzione progettuale ad un problema ricorrente. Nella costruzione di un player di qualsiasi tipo e in qualsiasi linguaggio è norma utilizzare il Pattern

Model-View-Controller (MVC). Utilizzando un Design Pattern un'applicazione diventa più semplice, robusta, flessibile e soprattutto riutilizzabile. Nel nostro caso il Pattern MVC specifica tre stati distinti e interagenti tra di loro:

- Model: sono i dati del programma e la logica di business applicata ad essi. E' compito del Model aggiornare i vari View associati ad esso.
- View: è la parte dell'applicazione visibile all'utente. Fornisce un'interfaccia con la quale l'utente può utilizzare e inserire dati.
- Controller: è lo strato dell'applicazione che unisce i due sopra descritti. Ha il compito di rilevare le azioni compiute dall'utente e di trasferirle al Model.

La tabella 5.2.4.1. raffigura l'interazione tra i vari componenti.

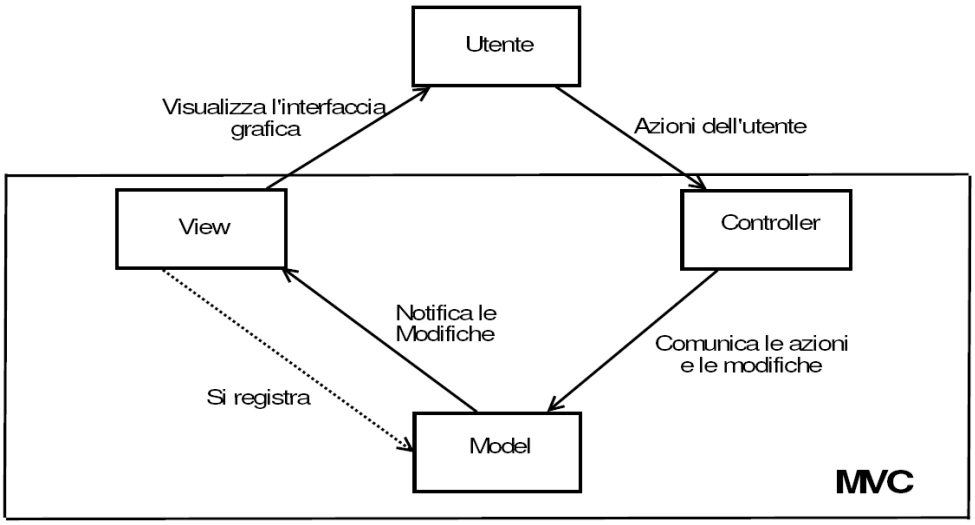


Tabelle 5.2.4.1 *Il Design Pattern Model View Controller*

Nel caso del player sviluppato, per ogni componente del Pattern è implementata da una classe Java distinta. Questa soluzione permette di applicare efficacemente il Pattern e sfruttarne le caratteristiche ma anche di tener conto dell'efficienza in un ambiente così limitato come i telefonini.

I vari componenti del Pattern svolgono compiti distinti ma devono essere sincronizzati. Le azioni dell'utente vengono riflesse sullo View

solamente quando è stato completato l'intero ciclo del MVC. In alcuni casi però il Model deve effettuare delle operazioni che richiedono molto tempo e che rendono l'usabilità dell'applicazione scarsa. E' il caso delle operazioni di I/O e cioè quelle operazioni che richiedono la lettura dei dati dalla memoria fisica. Nel caso del player questo tipo di operazioni è molto comune e l'accesso ai dati è abbastanza lento. Un'applicazione con i stati sincronizzati sarebbe inutilizzabile. Per risolvere questo problema si è scelto di utilizzare i Thread per le operazioni di I/O. Nel successivo capitolo verrà spiegata la struttura di un Thread e il suo utilizzo nell'applicazione.

5.2.5 L'utilizzo di Threads

Il Thread è una lista di istruzioni eseguita in modo sequenziale da un programma. I vari Threads di un programma condividono lo stesso spazio di indirizzamento ma hanno lo stack e le variabili locali proprie. Spesso vengono definiti come processi "leggeri" perché la loro creazione non è molto dispendiosa in termini di risorse. Il loro utilizzo permette di effettuare più operazioni in parallelo se non fanno utilizzo delle stesse risorse hardware e cioè se sono indipendenti. Esistono due modi per utilizzare un Thread:

- Estendento la classe Thread

  Esempio:

```
1  class MioThread extends Thread {
2      MioThread{
3      }
4      public void run() {
5         istruzioni
6         . . .
7      }
8  }
9
10 MioThread m = new MioThread();
```

```
11  m.start();
```

- Oppure implementando la classe Runnable

  Esempio:

```
1   class MioThread implements Runnable {
2       MioThread{
3       }
4       public void run() {
5         esegue qualcosa
6         . . .
7       }
8   }
9
10  MioThread m = new MioThread();
11  new Thread(m).start();
```

Entrambi i metodi devono sovrascrivere o implementare il metodo run() che verrà poi richiamato al momento dell'esecuzione del metodo start().

Nel caso del player, l'applicazione del Pattern MVC e l'indipendenza delle istruzioni di I/O dal resto del sistema permettono l'utilizzo di diversi Thread per i vari compiti. In particolare viene utilizzato un Thread sia nella creazione del Player sia nel caricamento delle immagini e dell'audio. In seguito verranno spiegati in dettaglio i casi di utilizzo:

- Creazione del Player: Nella creazione del Player tramite l'istruzione Manager.createPlayer(InputStream,ContentType) è possibile entrare in una situazione di I/O bloccante, cioè in cui l'applicazione attende per un lungo periodo la possibilità di utilizzare la risorsa desiderata. Utilizzando quest'istruzione in un Thread separato, l'applicazione non è bloccata nell'attesa e permette di eseguire altre operazioni mentre il Manager è in attesa di creare il Player.
- Caricamento dei dati da files: In questo caso ci sono due motivi per cui utilizzare un Thread separato. Il primo è per evitare le

situazioni di I/O bloccante come nel caso precedente mentre il secondo è per velocizzare il caricamento dei dati. Come descritto nel Capitolo 4.1.2 le immagini sono "raggruppate" in un unico file per risolvere i problemi dei permessi. In J2ME non esiste però un metodo di accesso diretto ad un file. In realtà è definito ma non è implementato correttamente. L'unico modo per accedere ad una determinata posizione è utilizzare il metodo InputStream.skip(n) che permette di "saltare" i primi n bytes e di posizionarci al (n+1)-esimo. Il problema principale è che l'istruzione skip(n) è inefficiente perché legge tutti i bytes dalla posizione corrente del puntatore fino a n e li scarta. Ogni accesso a immagini, pertanto, è vincolato dalla lentezza di accesso in modo sequenziale. La soluzione ideata per questo tipo di problema prevede l'utilizzo di un buffer di immagini precaricate e della memorizzazione della posizione del puntatore in modo da non ripartire sempre da 0 nel posizionamento all'interno del file.

- Memorizzazione della posizione corrente del puntatore: quando viene effettuata la lettura di un qualsiasi dato il puntatore all'interno del file rimane all'ultimo byte letto. Memorizzare la posizione quindi potrebbe essere utile in caso in cui si voglia leggere un dato successivo a tale posizione perché di dovrà "saltare" soltanto la differenza tra la posizione desiderata e la posizione attuale di bytes.
- Il buffer: è stato implementato come un array di immagini precaricate in memoria in modo da averle subito a disposizione quando si effettua un cambio di capitolo o di slide. Ovviamente il buffer non contiene tutte le immagini ma un numero limitato. Si verificano perciò situazioni in cui si effettua una richiesta di immagine che non è presente nel buffer. Le situazioni che si possono verificare (Immagine 5.2.5.1) sono le seguenti:

- Caso 1: si desidera visualizzare l'immagine già presente nel buffer. E' la situazione più semplice perché viene restituita subito l'immagine desiderata.
- Caso 2: si desidera visualizzare un immagine successiva non presente nel buffer. Si dovranno leggere le immagini saltando soltanto la differenza di bytes che separa l'ultimo byte letto da essa.
- Caso 3: si desidera un'immagina precedente e non presente nel buffer. Si dovranno leggere le immagini riposizionando il puntatore partendo da 0. Questo è il caso più dispendioso in genere.

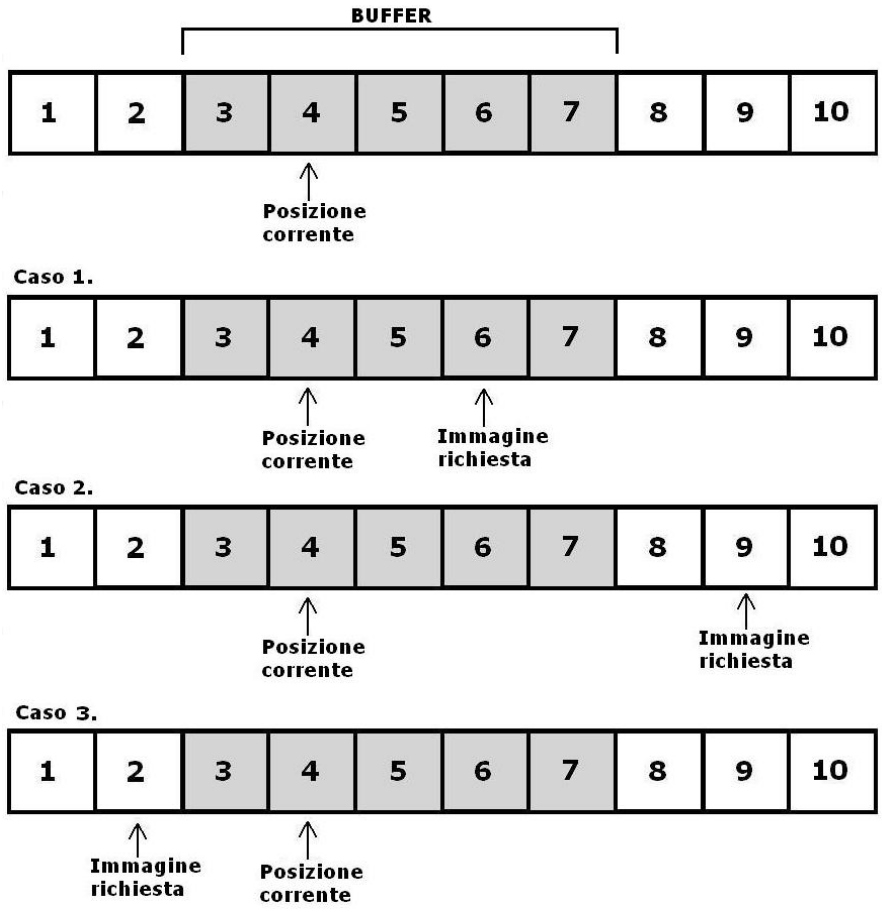


Immagine 5.2.5.1: I casi di accesso al buffer.

L’utilizzo dei Thread viene fatto per caricare il buffer in background, cioè mentre si sta visualizzando la presentazione. Le richieste nel caso 2 e 3 sono di tipo bloccante, cioè vengono esaudite immediatamente e l’applicazione nel frattempo è bloccata. Nello spostamento alla nuova posizione viene caricato anche il buffer perché i bytes letti, invece che scaricati, vengono memorizzati. Nel caso 1 invece, viene restituita l’immagine richieste e viene cambiata anche la posizione corrente. Mentre si sta visualizzando la presentazione il Thread incaricato legge le immagini successive e libera quelle precedenti in modo da riempire il buffer in modo corretto. Questa tecnica ha velocizzato di molto l’intera applicazione nel caso di caricamento parziale delle immagini.

- Zoom delle immagini: Un altro utilizzo dei Threads si ha nello zooming delle immagini. Come spiegato nel capitolo 4.2.1, nella scelta delle dimensioni delle immagini si è deciso di utilizzare una dimensione doppia dello schermo. Le immagini memorizzate quindi, per essere visualizzate per intere devono essere ridimensionate. La procedura che svolge questo compito è abbastanza efficiente ma tuttavia necessita di circa 300 millisecondi sul telefono utilizzato nello sviluppo per completare la procedura e visualizzare l’immagine. Avendo già a disposizione il buffer con le immagini precaricate si è scelto di utilizzare un altro buffer delle stesse dimensioni che contenga le immagini già ridimensionate. L’operazione di zooming viene effettuata dallo stesso Thread che carica le immagini da file.

5.3. Considerazioni generali

Le tecniche elencate nel capitolo precedente hanno permesso di risolvere gran parte dei problemi relativi all'accesso ai files. E' stato necessario però trovare un corretto rapporto tra la velocità di accesso ai dati e la quantità di memoria disponibile. Per quanto riguarda il buffer ad esempio si sono presentate due tipi di scelte da fare:

- La grandezza del buffer: il numero delle immagini precaricate non doveva essere troppo grande perché la quantità di memoria disponibile non era molta. Si devono caricare inoltre anche le tracce audio ed altri dati e quindi non si può utilizzare tutta la memoria disponibile.
- Distribuzione delle immagini attorno alla posizione corrente: la lettura delle immagini successive alla posizione corrente comporta un salto di bytes pari alla differenza delle due posizioni, mentre la lettura delle immagini antecedenti comporta un salto da 0 fino alla posizione dell'immagine desiderata. Bisogna trovare quindi un buon rapporto tra in numero delle immagini antecedenti alla posizione corrente da tenere nel buffer e il numero di quelle successive.

C'è da precisare però che tutto lo studio sul buffer e sulla sua utilità non è stato più necessario con il passaggio al secondo modello (Nokia N70) su cui è stato sviluppato il progetto, il quale aveva abbastanza memoria da contenere interamente tutte le immagini. Come già spiegato nei capitoli precedenti si è scelto di sviluppare una versione non compatibile con i modelli antecedenti e quindi anche il buffer delle immagini precaricate ha perso significato. La nuova versione dell'applicativo prevede solo il buffer delle immagini ridimensionate perché in questo modo si ha una navigazione molto più fluida tra una slide ed un'altra.

6. Implementazione

Nel seguente capitolo verranno spiegate in dettaglio le parti implementate sia per quanto riguarda l'automazione sia per quanto riguarda il player delle presentazioni. Verranno illustrati i casi d'uso e i vari diagrammi che descrivono l'architettura delle applicazioni. Verranno inoltre spiegate le parti più importanti con lo pseudo-codice.

6.1. Automazione e conversione

Come già spiegato in precedenza solo una parte dell'automazione è stata implementata in J2SE. Le restanti operazioni vengono effettuate con applicazioni di terze parti. Il loro utilizzo è spiegato in dettaglio nel capitolo 4.3. mentre in seguito verrà descritta la parte implementata.

6.1.1. Casi d'uso

I casi d'uso descrivono le modalità di utilizzo di un sistema. Nel nostro caso solo l'amministratore o chi gestisce la pubblicazione delle lezioni effettua la conversione. Nella figura 6.1.1.1. è rappresentato l'unico caso d'uso per quanto riguarda l'automazione della conversione. Come si può notare è un'operazione composta da vari passaggi.

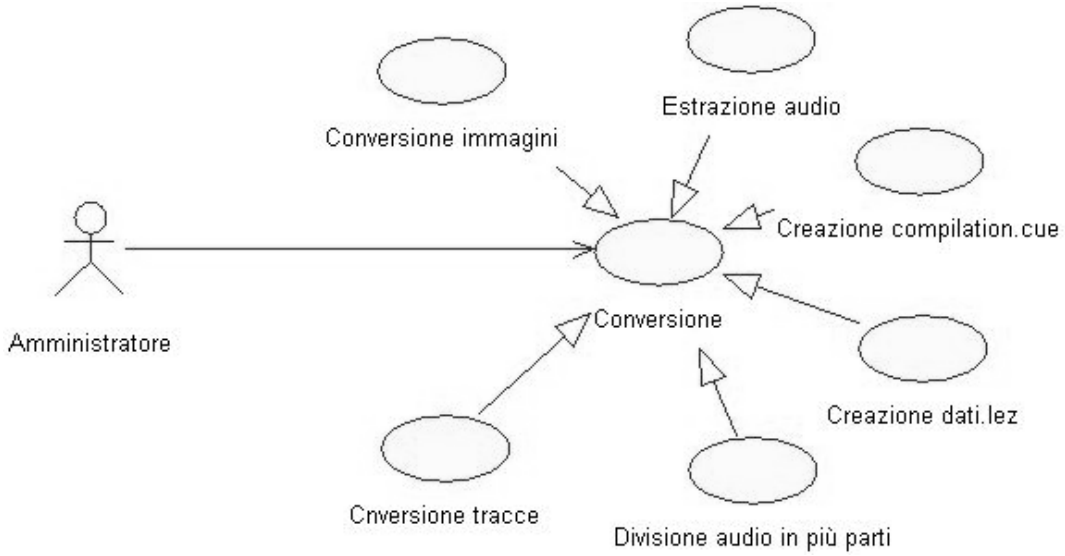


Figura 6.1.1.1: *Use case diagram*

6.1.2. Architettura

La procedura di automazione e della conversione, come già spiegato, è avviata da un file *Conversione.bat* che lancia in sequenza diverse applicazioni. L'architettura dell'applicazione è abbastanza semplice. Può essere descritta come una serie di manipolazioni e elaborazioni con vari programmi su una serie di dati che compongono una presentazione. Uno schema che rappresenta i vari passaggi è raffigurato in Figura 4.3.1.

6.1.3. Class Diagram

Il diagramma delle classi (class diagram) descrive le classi Java che compongono l'applicativo *GestioneArchivio.jar*.
Le varie classi:

- *Main*: istanzia un Archivio e richiama le funzioni principali.

- *Archivio*: è la classe che effettua le operazioni principali. Crea i files *compilation.cue* e *dati.lez* leggendo le informazioni dalla presentazione di partenza.
- *Capitolo*: è la classe che rappresenta un capitolo. La sua struttura è rappresentata nella Figura 4.1.2.3.

Il diagramma delle classi:

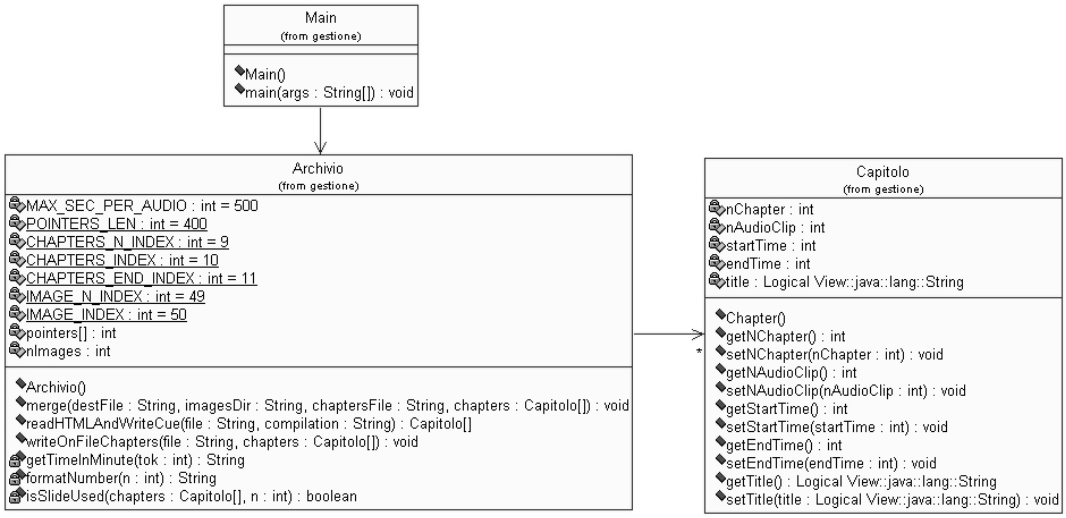


Figura 6.1.3.1: *Diagramma delle classi di GestioneArchivio*

6.1.4. Sequence Diagram

Il sequence diagram descrive l'ordine delle azioni che compie un programma. E' composto da una serie di attori che comprendono sia le persone che le classi del sistema e da un predefinita serie di azioni che essi svolgono.

Sequence diagram di *GestioneArchivio:*

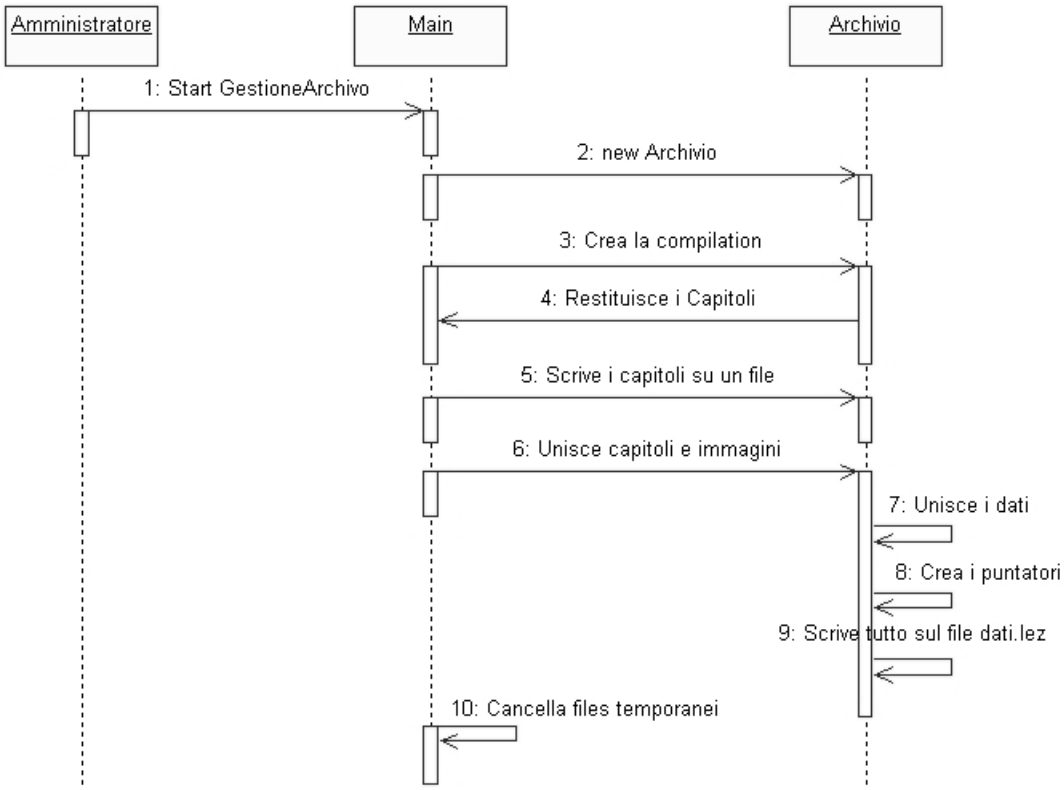


Figura 6.1.4.1: *Sequence Diagram di GestioneArchivio*

6.1.5. Pseudo-Codice delle parti importanti

Le parti più importanti che compongono GestioneArchivio sono:

- Creazione di compilation.cue:

```
1    istanzia l'array di capitoli
2    crea il file compilation.cue
3    apre file index.html
4    finché non trova "TICKS" "DEFINITION" salta una riga
5    se non trovata stampa errore
6    altrimenti per ogni riga (capitolo)
7       legge la riga
8       divide in tokens la riga
9       seleziona solo i tokens necessari
```

```
10      crea in capitolo e lo aggiunge all'array
11      se la lunghezza del capitolo supera il limite
12      impostato
13         scrive nel compilation.cue un nuovo punto dove
14         "spezzare" la traccia
15      altrimenti
16         legge un altro capitolo e lo unisce a quello
17         letto precedentemente
18      fine ciclo
19      restituisce l'array di capitoli al Main
```

Il punto più importante dell'algoritmo è la scelta dove interrompere una traccia audio e dove ricominciarne un'altra (riga 11). E' una scelta basata sulla disponibilità di memoria RAM dei dispositivi. Il limite impostato è di 500 secondi. Una traccia audio non dovrebbe superare il doppio del limite che corrisponde ad una traccia finale di circa un MegaByte ma non dovrebbe essere neanche inferiori della metà. In questo modo certe volte vengono raggruppati più capitoli in una traccia audio riducendo il numero delle richieste di permessi.

- Unione dei capitoli e delle immagini e creazione del file dati.lez:

```
1   crea un file temporaneo a.tmp
2   crea l'array dei puntatori
3   scrive in a.tmp i capitoli aggiornando i puntatori
4   scrive in a.tmp tutte le immagini aggiornando i
5   puntatori
6   scrive nel file destinazione i puntatori
7   ne misura la lunghezza
8   aggiorna i puntatori sommando la lunghezza misurata
9   precedentemente
10  riscrive nel file destinazione i puntatori aggiornati
11  aggiunge al file destinazione il contenuto del file
12  a.tmp
```

E’ stato necessario effettuare i passaggi nella riga 6 e 7 perché i puntatori erano riferiti ad un file contenente solo capitoli e immagini. In esso però si dovevano aggiungere anche i puntatori all’inizio. In questo modo però i puntatori non sarebbero stati più corretti perché i capitoli non sarebbero più stati nella posizione iniziale del file. Una soluzione è stata scriverli in un file, misurarne la lunghezza e sommarla ad ogni puntatore. In questo modo si riottenevano riferimenti corretti.

6.2. Player

Come già spiegato nel Capitolo 5.1 l’ambiente di sviluppo utilizzato è stato NetBeans 5.5.1 IDE con supporto di WTK 2.5.1 per quanto riguarda le librerie CLDC e MIDP e Carbide.j per l’emulazione. L’applicativo ottenuto dalla compilazione del progetto può essere installato tramite il software fornito assieme al telefonino con un paio di semplici passaggi. Nei capitoli successivi verranno spiegati in dettaglio la struttura del player e il suo utilizzo.

6.2.1. Casi d’uso

I casi d’uso del player sono tutte le funzionalità che offre nella visualizzazione delle lezioni. Tutti gli agenti possono svolgere le stesse attività e sono rappresentati sotto il nome Utente.

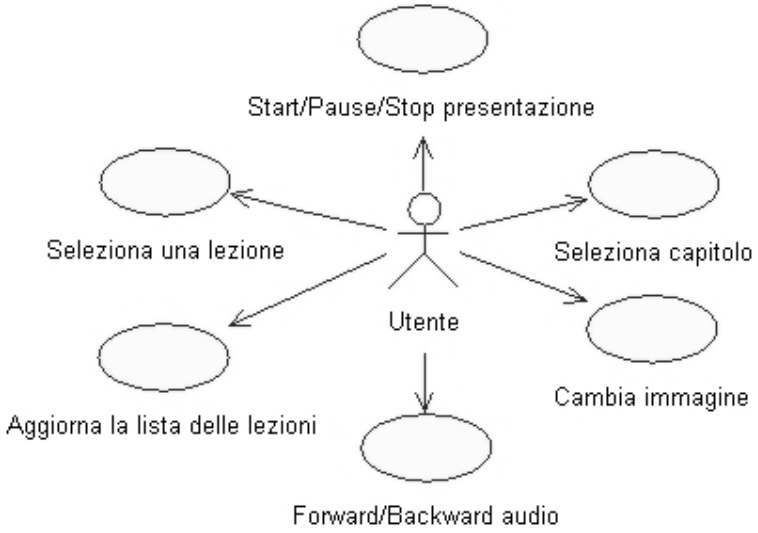


Figura 6.2.1.1: *Casi d'uso del player.*

6.2.2. Architettura

L'architettura dell'applicazione, come già spiegato nei capitoli precedenti, segue il più possibile il Pattern Model-View-Controller. Il player dispone di due Interfacce Utente. Una è fornita dal LODEPlayer che permette di scegliere la lezione da visualizzare e una dallo View e cioè la presentazione con immagini ecc. In J2ME l'interfaccia utente può essere rappresentata come una lista di oggetti selezionabili. Ad ogni lista è associata una lista di comandi. C'è un conflitto quindi per quanto riguarda il View e il Controller del Pattern MVC perché entrambi fanno parte di una stessa classe. Siccome le operazioni associate ai vari comandi sono abbastanza semplici si è scelto di non complicare ulteriormente la struttura anche a beneficio dell'efficienza e quindi di raggruppare il View e il Controller in una stessa classe. Il Data Manager in figura 6.2.2.1 rappresenta il Model del Pattern MVC. Esso gestisce i dati di una lezione con la logica di Business spiegata nel capitolo 5.2.

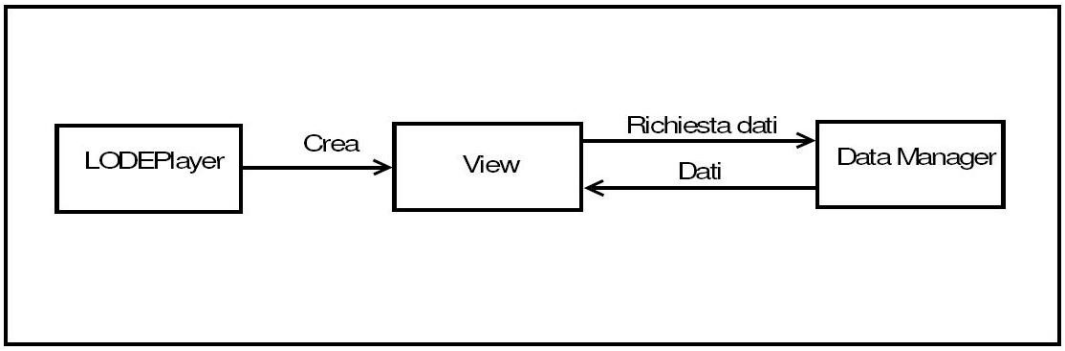


Figura 6.2.2.1: *Architettura del player*

6.2.3. Class Diagram

Il diagramma delle classi rappresenta la struttura e il tipo di iterazione che hanno le classi Java sviluppate. Le classi principali sono:

- LODEPlayer: è la classe che gestisce la lista delle lezioni disponibili in memoria e il lancio delle presentazioni. Quando viene selezionata una lezione istanzia il PlayerCanvas e fa partire il Player.
- PlayerCanvas: è la classe che gestisce una presentazione. Implementa la classe Runnable per la gestione del I/O bloccante del Player. Gestisce tutti i comandi e permette la selezione dei capitoli. Istanzia l'Archivio passando come parametro il path della presentazione scelta.
- Archivio: carica in memoria le informazioni riguardanti i puntatori, capitoli e le immagini. Implementa la classe Runnable per la gestione del buffer delle immagini e delle immagini ridimensionate come già spiegato nel capitolo 5.2. Ha il compito di fornire le immagini al PlayerCanvas quando ne fa richiesta.
- ErrorScreen: estende la classa Alert che permette di mostrare un avviso per una certa durata di tempo. Viene utilizzata per mostrare gli errori in caso ce ne fossero.

- Capitolo: Viene utilizzata nell'Archivio quando viene letto un insieme di capitoli. Contiene tutte le informazione sui tempi di cambio tra le slide e tra i vari capitoli.

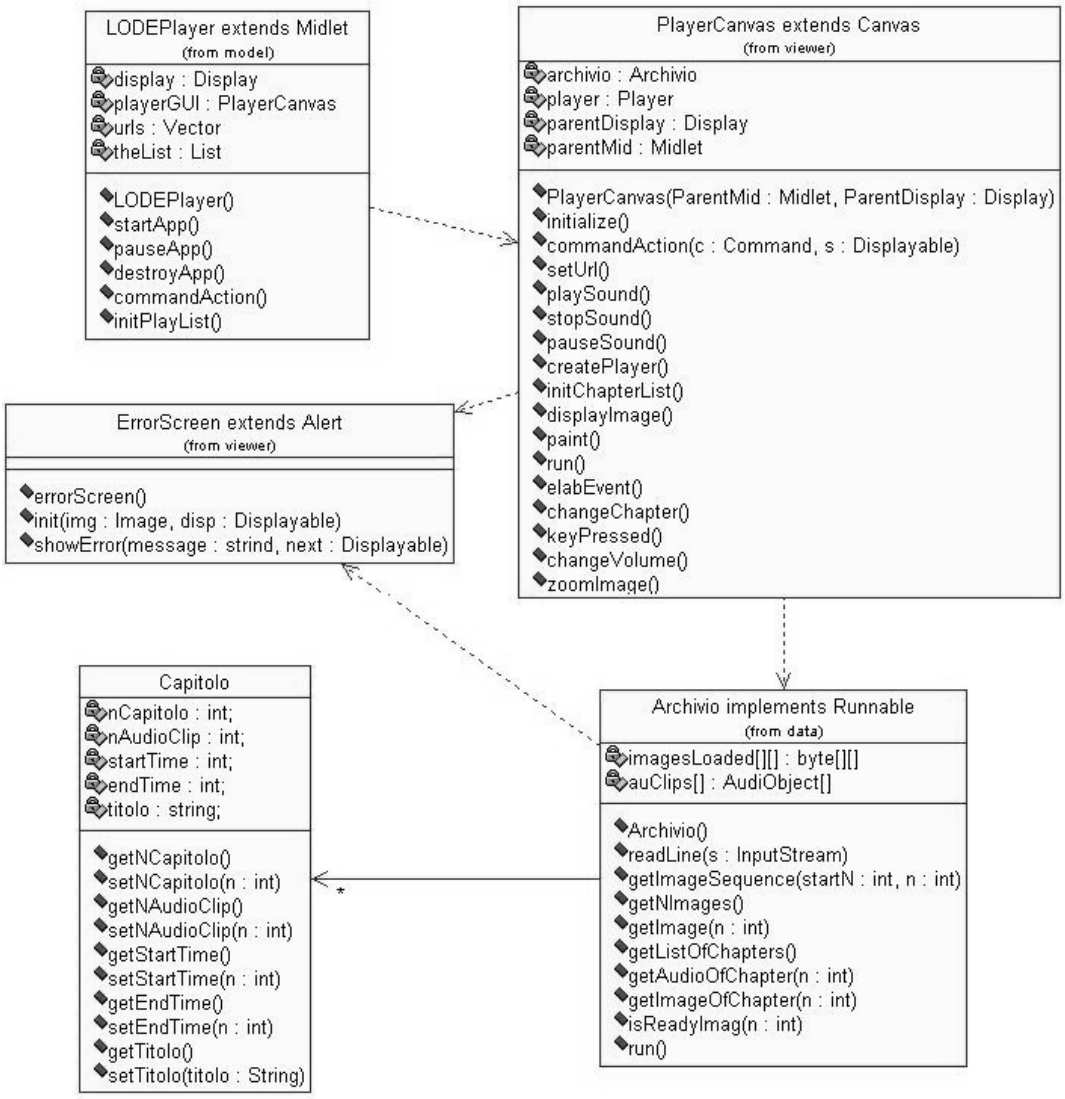


Figura 6.2.3.1: *Diagramma delle classi del player*

6.2.4. Sequence Diagram

Il seguente sequence diagram rappresenta il lancio dell'applicazione e la selezione di una presentazione da visualizzare. Si può notare in che ordine vengono lanciati i vari Thread e come il caricamento delle immagini nell'Archivio prosegua mentre il PlayerCanvas crea il Player e lo fa partire. Da notare che se l'immagine richiesta non è stata ancora caricata l'applicazione attende fino a che non viene fornita.

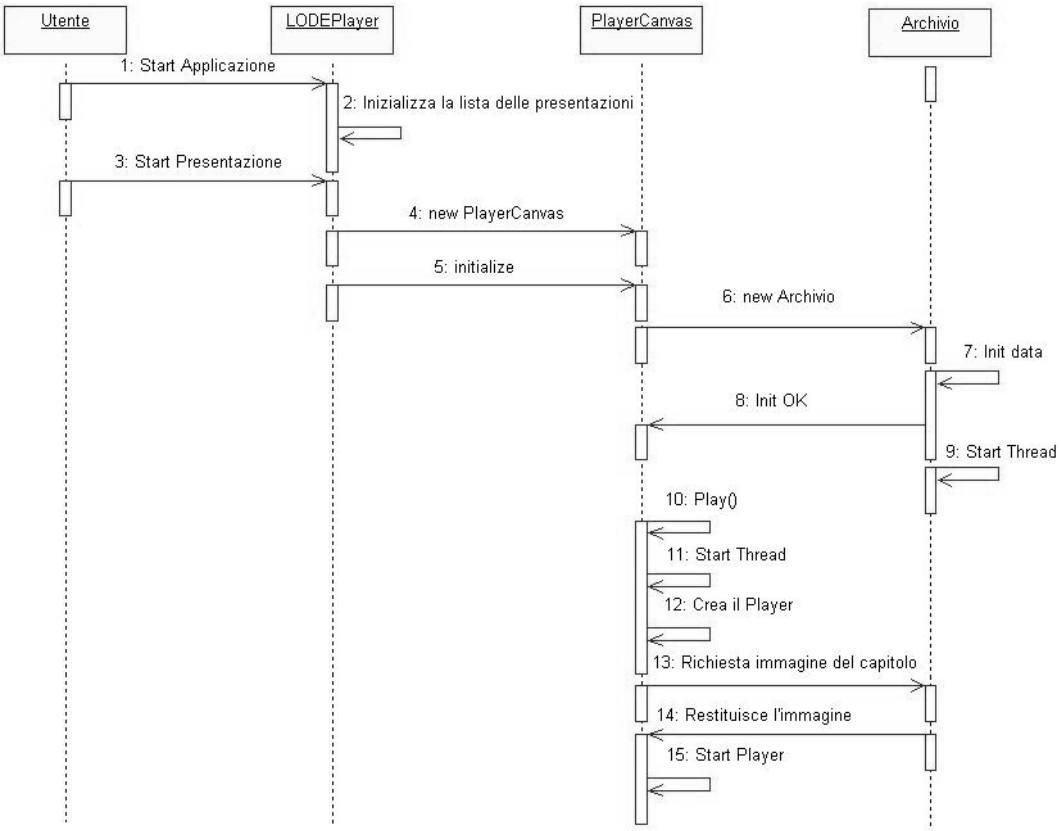


Figura 6.2.4.1: *Sequence diagram del player*

6.2.5. Interfaccia Utente

La realizzazione dell'interfaccia grafica la quale consente di interagire con l'utente è una parte fondamentale nell'ambito della programmazione MIDP. L'usabilità dell'applicazione, uno dei principali requisiti del progetto, dipende molto da essa. In MIDP 2.0 sono presenti due tipi di API per la gestione dell'interfaccia grafica (GUI):

- API di alto livello (high level API)
- API di basso livello(low level API)

La principale differenza tra le due tipologie di API riguarda la portabilità. Le API di alto livello adottano un modello simile alle API AWT di J2SE, le quali descrivono alcuni strumenti con i quali l'utente può interagire con l'applicazione delegando totalmente la gestione al particolare dispositivo. Le API di alto livello verranno utilizzate nella gestione dei menù e delle liste delle lezioni o dei capitoli. Per creare una lista selezionabile basta infatti creare un'istanza della classe List, aggiungere gli elementi che si desiderano e impostarla come oggetto da visualizzare con il commando Display.setCurrent(List). L'oggetto List verrà quindi visualizzato diversamente su vari dispositivi con caratteristiche differenti.
Le API di basso livello permettono invece di aver un maggior controllo e gestione dell'aspetto estetico degli oggetti che si vogliono visualizzare. Offrono funzioni per disegnare direttamente a schermo figure o immagini e vengono utilizzate maggiormente nella creazione di giochi. Richiedono però maggior attenzione nella programmazione perché si deve tener conto delle diverse caratteristiche dei dispositivi, soprattutto per quanto riguarda la dimensione dello schermo. Nell'applicazione sviluppata vengono utilizzate nella visualizzazione delle immagini della presentazione e nella schermata di scorrimento dell'audio. Nel primo caso si fa uso di alcune funzione destinate alla

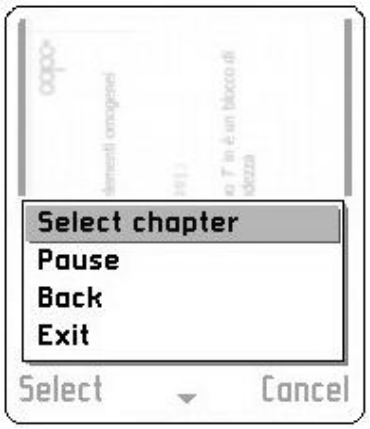
manipolazione delle immagini, mentre nel secondo gli oggetti vengono disegnati direttamente sullo schermo grazie all'oggetto Graphics. In seguito verranno mostrate le principali schermate che si presentano durante l'utilizzo del player:

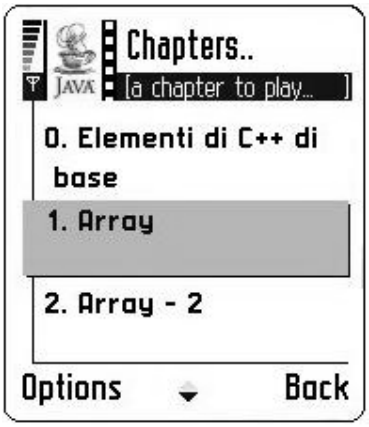
E' la schermata iniziale. Rappresenta l'elenco delle lezioni disponibili sul dispositivo. Al primo lancio dell'applicazione non sono presenti lezioni finché non si effettua una ricerca. Si può notare il primo esempio di utilizzo di una List che contiene la lista delle lezioni. La possibilità di scelta delle varie lezioni viene fornita direttamente dalle API.

Premendo il tasto corrispondente ad Options si accede al menù della prima schermata. È possibile scegliere tra Play che fa partire la presentazione selezionata, Search lections che aggiorna la lista delle presentazione sul dispositivo o Help che mostra la schermata di aiuto elencando i vari tasti utilizzabili.

Questo è il tipo di schermata che si presenta quando si tenta di accedere alle funzionalità delle API con restrizioni di privilegi. Questo tipo di "pop-up" si presenta solamente se l'applicazione non è firmata (come nel nostro caso).

Quando è in esecuzione una presentazione è possibile entrare nel menù premendo il tasto corrispondente a Select. È possibile selezionare un capitolo scegliendo Select Chapter, mettere in pausa la presentazione, tornare indietro scegliendo Back oppure uscire completamente scegliendo Exit.

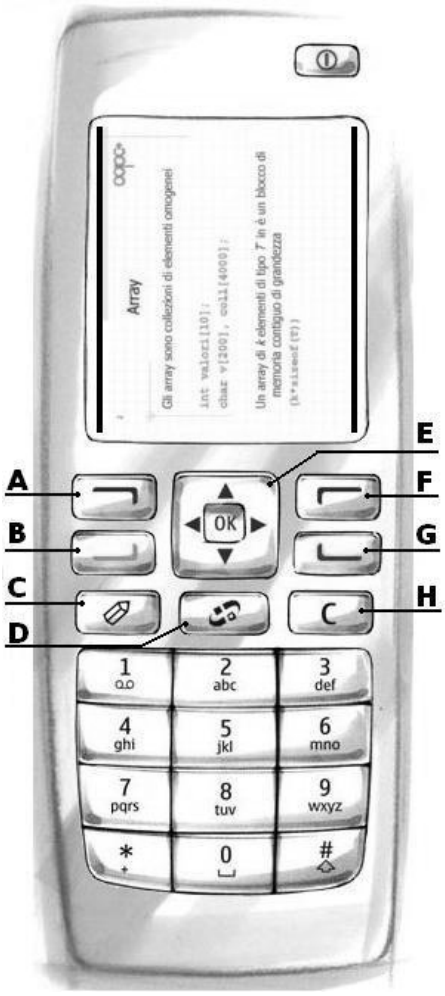




Se dal menù precedente viene selezionata l'opzione Select chapter, viene mostrato l'elenco di tutti i capitoli della presentazione. È possibile selezionare il capitolo che si desidera oppure tornare indietro alla presentazione. Selezionando alcuni capitoli non verranno chieste autorizzazioni di accesso ai dati perché l'audio corrispondente è all'interno della stessa traccia di quello che si stava già consultando.

Nell'immagine successiva è raffigurata una tastiera standard dei telefonini classici, cioè non del tipo QWERTY[31]. Si dispone di 12 tasti numerici (compresi * e #) e di altri tasti associati a funzioni di sistema e/o navigazione. I tasti numerici possono essere utilizzati da qualsiasi applicazione sviluppata in MIDP 2.0. Gli altri tasti possono essere utilizzati solo parzialmente. Le loro funzionalità sono:

[31] http://it.wikipedia.org/wiki/Tastiera_(informatica)



- A: è il tasto che corrisponde di solito ad Options. Viene utilizzato per accedere al menù associato al display.
- B: tasto di chiamata. Non può essere utilizzato dall'applicazione.
- C: tasto copia. Non può essere utilizzato dall'applicazione.
- D: tasto menù. Serve per accedere al menù del telefono o per mettere in background l'applicazione corrente.
- E: il Joystick. Comprende 4 tasti direzionali e il tasto di conferma centrale. Quest'ultimo viene utilizzato per dare conferma nei vari menù. I tasti direzionali possono essere utilizzati da qualsiasi applicazione MIDP ma non in modalità FullScreen. La visualizzazione delle immagini è effettuata in modalità FullScreen e quindi i tasti direzionali non possono essere utilizzati.
- F: tasto Back. Di solito ad esso è associata l'uscita dall'applicazione o il ritorno alla schermata precedente.
- G: tasto Exit. In alcuni modelli premendo tale tasto si ritorna al menù principale del telefonino mettendo in background l'applicazione in uso. In altri l'applicazione viene chiusa completamente.
- H: tasto Cancel. Viene utilizzato per cancellare del testo scritto. Non può essere utilizzato dall'applicazione.

I tasti numerici vengono utilizzati per tutte le funzioni associate alla navigazione attraverso le presentazioni. La scelta dei tasti con cui effettuare le varie operazioni influenza molto l'usabilità del sistema. Le presentazioni vengono consultate tenendo il dispositivo in modo orizzontale a causa della forma delle immagini. Le funzionalità associate ai tasti sono raffigurate nell'immagine 6.2.5.1 e sono le seguenti:

- 1,3: Permettono di visualizzare rispettivamente l'immagine successiva e precedente.
- 2,4,6,8: Sono tasti direzionali. Spostano la visualizzazione dell'immagine rispettivamente in su, a destra, in giù e a sinistra.
- 5: Effettua l'ingrandimento dell'immagine se è visualizzata interamente, altrimenti la ridimensiona.
- 7,9: Scorrimento veloce della traccia audio rispettivamente in avanti e indietro.
- *,#: rispettivamente aumenta e diminuisce il volume.

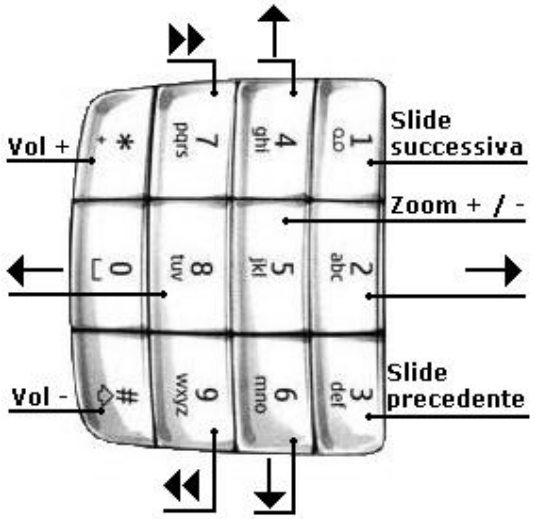


Figura 6.2.5.1: *La tastiera e le funzioni principali*

6.2.6. Pseudo-Codice delle parti importanti

In seguito verranno illustrati i passaggi principali sia della compilazione e installazione del player sia quelli implementativi.
Compilando il progetto si ottengono due files:

- LODEPlayer.jar: è l'archivio contenente tutte le classi JavaME ed eventuali immagini e audio incorporati nel progetto. L'applicazione può accedere a tutte le informazioni contenute nel file jar. Per qualsiasi altro accesso ci sono restrizioni di privilegi.
- LODEPlayer.jad: Java Application Descriptor, è il file di descrizione del LODEPlayer.jar. Contiene alcune informazioni riguardanti il nome dell'applicazione, l'icona, la dimensione, informazioni sul venditore e le versioni di CLDC e MIDP utilizzate.

I due files sono necessari per l'installazione del player che può essere effettuata sia scaricando i files da internet che installandoli con l'apposito software fornito dalla casa produttrice de dispositivo.
In seguito verranno illustrate le principali tecniche adottate nella programmazione delle varie classi JavaME.

- LODEPLayer: i due compiti principali della classe sono: gestire la lista delle presentazioni disponibili e il lancio del Player. La prima operazione fa uso della memoria persistente fornita dalle MIDlets per memorizzare le presentazioni. Tale memoria simula un database e si chiama RMS (Record Managment System). Il funzionamento della gestione della lista delle presentazione è spiegato dal seguente pseudo-codice:

```
1 se il record non esiste (primo lancio)
2   crea il record
3   legge in memoria le presentazioni disponibili
4   le aggiunge al record
5   salva il record
6 crea l'oggetto List contenente le presentazioni
7 imposta List come display
```

Il lancio della presentazione selezionata avviene quando è selezionato un oggetto dalla lista:

```
1 selezione di una presentazione dalla lista
2 ottiene il path della presentazione
3 crea PlayerCanvas
4 imposta il path della presentazione da riprodurre
5 importa il PlayerCanvas come display
6 inizializza il PlayerCanas
```

- PlayerCanvas: i principali compiti della classe sono la creazione dell'oggetto Player e la visualizzazione delle immagini.
  Il Player viene creato nel metodo run() del Thread implementato per non creare situazioni di I/O bloccante. Lo pseudo-codice del metodo run() è il seguente:

```
1 crea il Player richiamando l'istruzione
2 Manager.createPlayer(InputStream,ContentType)
3 effettua il Prefetch del Player
4 effettua il posizionamento nel audio al tempo
5 corrispondente all'inizio del capitolo
6 lancia il Player
7 aggiunge il PlayerListener al Player
8 imposta il tempo di fine del capitolo
```

  Un'altra operazione importante implementata nella classe PlayerCanvas è il cambio del capitolo. Esso si verifica sia quando è finito il capitolo corrente sia quando viene selezionato un nuovo capitolo dal menù scelta capitoli. Lo pseudo-codice della gestione del cambio dei capitoli è:

```
1 imposta l'immagine di attesa
2 ferma il Player
```

```
3  ottieni il path dell'audio associato al capitolo
4  selezionato
5  se l'audio selezionato è uguale a quello corrente
6    effettua il riposizionamento all'interno della
7    traccia
8  altrimenti
9    crea il Player associato alla nuova traccia audio
10   effettua il Prefetch del Player
11 mostra l'immagine associata al capitolo
12 fa partire il Player
```

Da notare che le istruzioni nella riga 10 e 11 sono fuori dalla condizione in riga 5. Nella riga 8 viene ripetuta sostanzialmente la procedura di lancio iniziale del Player. E' un operazione abbastanza onerosa e richiede del tempo sia per la lettura della traccia sia per il suo lancio.

- Archivio: il compito principale della classe Archivio è la lettura delle informazioni riguardanti i capitoli di una presentazione e la lettura delle immagini.

  La lettura dei capitoli e puntatori:

```
1  apre il file dati.lez
2  legge i puntatori e li salva in un array
3  legge il numero N dei capitoli presenti
4  dalla posizione iniziale dei capitoli legge tutti i
5  capitoli e li aggiunge in un array di dimensione N
6  legge il numero delle immagini presenti M
7  legge tutte le immagini e le salva in un array di
8  dimensione M
```

  Gli indici sono memorizzati come numeri interi uno per ogni riga nel file dati.lez. La lettura consiste in un ciclo che li legge tutti effettuando un parsing da stringhe a interi. I capitoli sono memorizzati anch'essi sotto forma di stringhe sequenziali. Il

metodo di lettura è simile ed effettua i vari parsing a seconda del campo letto.

Il thread che effettua lo zooming delle immagini in background è un'altra funzionalità importante della classe Archivio. Il suo funzionamento è il seguente:

```
1 ogni 4 secondi:
2   se c'è stata una richiesta di immagini nell'ultimo
3   periodo
4     effettua lo zoom delle immagini prossime
5     all'immagine richiesta
6     salva le immagini ottenute nel buffer.
```

Il suo funzionamento è definito nella funzione run() della classe. Le controlli vengono effettuati ciclicamente ogni 4 secondi ma questa è una scelta che deve essere fatta per non impegnare per troppo tempo la CPU e può essere reimpostata. La metodologia di gestione del buffer è spiegata nel capitolo 5.2.5 e tiene conto sia dei tempi di riempimento sia della memoria occupata.

7. Conclusioni

I risultati ottenuti sono stati soddisfacenti sotto certi punti di vista e meno sotto altri. Sono state implementati tutte le funzionalità contenute nei requisiti funzionali non senza qualche difficoltà. C'è da considerare che senza i problemi imposti delle scelte relative alla sicurezza i risultati sarebbero stati ottimi. L'usabilità dell'intera applicazione è stata condizionata dai metodi utilizzati per aggirare le regole di sicurezza imposte dalle API MIDP e dal Symbian OS. Le caratteristiche hardware dei dispositivi sarebbero più che sufficienti in un ambiente un po' più "libero". Una conclusione che può esser tratta dallo studio è che pur disponendo di risorse hardware quasi superiori ai personal computer come i primi Pentium I, il sistema operativo non riesca a sfruttare tutte le sue caratteristiche. Si ha l'idea che il "collo di bottiglia" stia proprio nel sistema operativo e nelle funzionalità offerte dal *profile* MIDP 2.0. Utilizzando altri linguaggi di programmazione, come ad esempio C++, si avrebbero sicuramente dei vantaggi in termini di prestazioni e potenzialità ma anche dei svantaggi come ad esempio la portabilità o la complessità i programmazione.
Nei seguenti capitoli verranno riassunte le varie difficoltà incontrate e i possibili sviluppi futuri.

7.1. Difficoltà incontrate

Come già spiegato nei capitoli precedenti il risultato finale è stato condizionato dalle specifiche tecniche dei dispositivi (CPU, RAM ecc.) e dalle funzionalità offerte dalla piattaforma di sviluppo (Symbian e MIDP).

In seguito saranno riassunte le principali difficoltà incontrate suddivise in due categorie, quelle che riguardano l'hardware e quelle che riguardano il software dei dispositivi:

- Limitazioni delle specifiche tecniche dei dispositivi:
  - La memoria fisica dei dispositivi è molto limitata. Di conseguenza c'è bisogno di estenderla con delle memory card più capienti di quelle in dotazione. Non ha rappresentato un limite nello sviluppo.
  - La memoria RAM dei dispositivi prodotti antecedentemente a quello di sviluppo è abbastanza ridotta. Si è dovuto implementare dei metodi per limitarne l'utilizzo come ad esempio la gestione dei buffer.
  - La velocità della CPU è sufficiente ma la velocità di lettura dei dati dalla memoria fisica non permettono implementazioni di ulteriori funzionalità come ad esempio la visualizzazione del video delle lezioni.
- Limitazioni dell'ambiente di sviluppo: i limiti maggiori sono stati riscontrati nelle caratteristiche e funzionalità offerte dall'ambiente di sviluppo J2ME.
  - I problemi maggiori si sono avuti con le restrizioni relative alla sicurezza. L'accesso ad API "sensibili" (Capitolo 2.3.5) è vincolato dall'acquisizione del consenso da parte dell'utente in caso in cui l'applicazione non sia firmata. Non possedendo certificati validi si sono dovuti implementare meccanismi per evitar le richieste di accesso. In seguito però si sono presentati problemi relativi a tali meccanismi. Essi saranno descritti nei punti successivi.
  - Un altro problema non meno importante è stato causato dall'assenza di librerie per l'accesso diretto a files. Una conseguenza del problema descritto nel punto precedente è stata l'implementazione di un archivio proprio che contenga

tutte le immagini. Un meccanismo del genere ha risolto i problemi relativi ai permessi ma non avendo modo di accedere direttamente ai files si son dovuti implementare dei buffer per leggere in anticipo le informazioni minimizzando i tempi di attesa per il riposizionamento sequenziale. Questo ha influito molto sull'usabilità dell'applicazione.

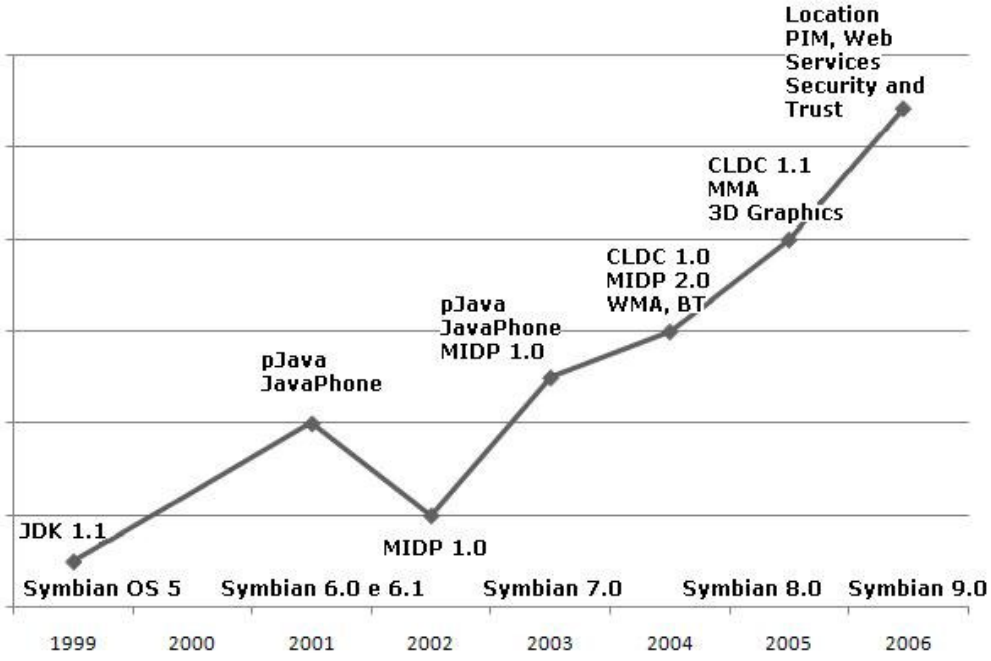
- Sono stati inoltre riscontrati alcuni errori di implementazione di alcune API, che sono stati anche riconosciuti ufficialmente dalla Sun Microsistems.

  Quello di maggior impatto è il malfunzionamento della funzione `Player.setMediaTime(long n)` che solo su alcuni modelli con la versione del *firmware* aggiornato permette il posizionamento all'interno di una traccia audio al punto desiderato. Questo difetto impedisce il funzionamento del player su modelli "difettosi" perché la funzione viene richiamata spesso durante l'esecuzione.
- Un altro problema non meno importante è la non efficacia della *Garbage Collection*[32] di J2ME che permette sostanzialmente di recuperare lo spazio di memoria non più utilizzato. Dovendo caricare in memoria molte immagini e audio relativamente grossi si ha la necessità che la memoria sia liberata. Impostando ogni riferimento ad un oggetto a *null* la memoria non viene liberata come accade in Java SE. Inserendo richiami "forzati" al metodo `System.gc()` c'è un miglioramento ma il problema non viene risolto definitivamente. Il problema diventa critico nei dispositivi con ridotta memoria.

[32] http://it.wikipedia.org/wiki/Garbage_collection

7.2. Sviluppi futuri

Per quanto riguarda sviluppi futuri si aprono molti spiragli e possibilità sia dal punto di vista di nuove funzionalità in ambiente J2ME sia per quanto riguarda la comparsa sul mercato di nuovi dispositivi con sistemi operativi diversi e potenzialità di sviluppo diverse come ad esempio l'iPhone o Android[33] di Google. Per quanto riguarda i nuovi dispositivi e la loro diffusione non è possibile prevederne la diffusione nei prossimi anni. Al momento attuale Nokia detiene circa il 40% del mercato per quanto riguarda la vendita di telefonini. Non è possibile prevedere quindi neanche l'utilità di uno sviluppo di applicazioni per il mobile learning. Si possono fare però delle considerazioni per quanto riguarda l'evoluzione del progetto in ambiente in cui è stato sviluppato, vale a dire J2ME. Come si può vedere dal grafico 7.2.1 l'evoluzione delle funzionalità in ambiente J2ME segue lo sviluppo del Symbian OS e ne sta diventando sempre di più una parte integrante.



[33] http://www.googleandroid.org/

Grafico 7.2.1: *Evoluzione delle funzionalità in J2ME.*

Il numero delle funzionalità offerte e degli standard (JSR) crescono in modo esponenziale seguendo le potenzialità hardware e del Symbian OS. Come si può notare dal Grafico 7.2.2 anche la velocità di esecuzione delle librerie MIDP 2.0 sta crescendo in modo significativo.

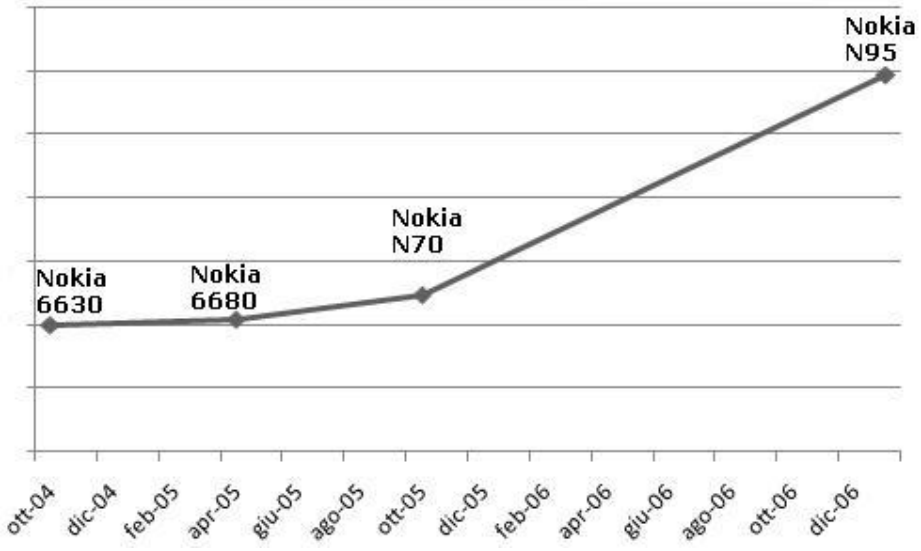


Grafico 7.2.2: *Velocità di esecuzione delle librerie MIDP 2.0* [JBC]

Un altro tipo di sviluppo che si sta verificando riguarda la connettività dei telefonini. Quasi la totalità possiede diversi tipi di connettività tra le quali GPRS ed UMTS, ma ultimamente si stanno diffondendo tecnologie che permettono velocità di navigazione più elevate come ad esempio HSDPA. Tale tecnologia permette il download dei dati ad una velocità massima di 7.2 Mbit/s. Molte compagnie telefoniche offrono la visione delle partite di calcio tramite i telefonini di ultima generazione. Si prevede anche compatibilità con lo standard WiMax, che in un futuro abbastanza breve coprirà tutto il territorio nazionale offrendo connessioni a velocità alta a prezzi ridotti. Un tale tipo di connessione potrebbe fornire un accesso real-time ai contenuti disponibili in rete. La versione di Symbian OS sulla quale è stato

sviluppato il player non dispone di librerie per la visione dei contenuti real-time, ma le versioni successive si. Si può ipotizzare quindi una diffusione sempre maggiore di dispositivi che siano in grado di sfruttare tutte le tipologie di connettività, soprattutto se si tratta di connessioni a basso costo e con modalità flat. In un tale contesto si avrebbe sicuramente una maggior utilità e sviluppo di applicazioni di mobile learning.

8. Bibliografia

[ADW] David Fox, Roman Verhosek - Programming Wireless Devices With The Java 2 Platform, Micro Edition, Second Edition - Addison Wesley 2002.

[JBC] jBenchmark: http://www.jbenchmark.com/index.jsp

[JOW] Martin de Jode - Programming Java 2 Micro Edition for Symbian OS – Wiley 2004.

[LODE] Ronchetti, M. (2003). Using the Web for diffusing multimedia lectures: a case study. Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2003

[NF1] MIDP: Mobile Media API Support In Nokia Devices, http://forum.nokia.com

[NKF] Nokia forum and discussion board, http://www.forum.nokia.com/

[MXC] Massimo Carli, Un libro gratuito su MIDP 2.0, http://www.massimocarli.it/site/

[THO] P. Thornton, and C. Houser, (2006) "Using mobile phones in education", in Proceedings of the 2nd IEEE International Workshop on Wireless and Mobile Technologies in Education, 2006, pp. 3

[TRI] Trifonova A., Georgieva E., Ronchetti M., (2006) "Has the Time for University's Mobile Learning Come?". WSEAS Transactions on Advances in Engineering Education, 2006, v. 3, n. 9