ALBERTO GUIOTTO

Matr. 563177

Laurea Triennale in Ingegneria Elettronica

Relatore: Prof. LORENZO VANGELISTA

“Descrizione dell’ambiente di sviluppo per applicazioni Symbian S60 con applicazione di test”

18 / 02 / 2010

ANNO ACCADEMICO 2009/2010

INDICE

Sommario . . . . . pag. 4

Cos'è Symbian OS . . . . . pag. 5

Cenni sulla storia . . . . . pag. 5

Caratteristiche principali di Symbian OS . . . . . pag. 6

Struttura del Sistema Operativo . . . . . pag. 11

Piattaforme per Symbian . . . . . pag. 13

S60 Fifth Edition . . . . . pag. 15

Linguaggi per sviluppo software in Symbian S60 Fifth Edition . . . . . pag. 18

Symbian C++ . . . . . pag. 18

Qt . . . . . pag. 19

Web Runtime (WRT) . . . . . pag. 19

Flash Lite . . . . . pag. 21

Python . . . . . pag. 22

Java ME su Symbian S60 Fifth Edition . . . . . pag. 23

Implementazione applicazione "Hello world" . . . . . pag. 31

SOMMARIO

Il testo ha lo scopo di descrivere l'ambiente di sviluppo per applicazioni destinate al sistema operativo Symbian S60. Vengono introdotte le nozioni base riguardanti la storia e la provenienza del sistema operativo, analizzandone successivamente le caratteristiche e spiegandone le utilità, per poi passare ad esaminarne la struttura dal punto di vista dei "*layers*".
L'analisi si sposta poi sulle piattaforme costruite per tale OS, scorrendo brevemente la cronologia delle varie edizioni; un focus viene fatto sulla nuova edizione "S60 Fifth Edition", illustrando le innovazioni e le tecnologie adottate. Rimanendo nell'ambito di tale piattaforma, si descrivono i linguaggi utili per sviluppare applicazioni dedicate a vari ambiti, complete degli strumenti per l'implementazione.
Un capitolo a parte è dedicato al linguaggio Java ME, nel quale sono spiegati in dettaglio le tecnologie e i metodi per la creazione di applicativi Java e il loro impiego.
Nell'ultima parte, si analizza in dettaglio la costruzione di un applicativo di test in stile "Hello World", completo di listato e analisi dei singoli metodi.

COS'È SYMBIAN OS

Symbian è un sistema operativo integrato per dispositivi mobili e smartphone. È sviluppato solamente per processori ARM, eccezion fatta per una release mai presentata per l'architettura x86.

Tale ambiente possiede proprie librerie associate, una User Interface caratteristica per ogni piattaforma, implementazioni di riferimento per strumenti comuni (agenda, browser web, etc.) e *framework*s per lo sviluppo software.

Il sistema operativo non è attualmente open source, anche se quasi tutto il codice è fornito agli sviluppatori di dispositivi mobili; inoltre le API (Application Programming Interface) sono rese pubbliche cosicché chiunque possa creare software per questa piattaforma.

Attualmente Symbian detiene il 50% del mercato smartphone, ed è considerata una leader del settore.

L'ultima versione disponibile di tale sistema operativo è la v9.5, che include, fra le altre peculiarità, il supporto nativo per il segnale DVB-H, comunemente chiamato Digitale Terrestre.

Cenni sulla storia

Symbian deriva dal sistema operativo EPOC di Psion. Quest'ultima nasce nel 1980, dando origine alla famiglia di Sistemi Operativi Grafici (detti anche GUI, ossia Graphic User Interface) per dispositivi portatili. L'acronimo EPOC deriva dal sostantivo "epoch" (termine inglese che significa "epoca"); successivamente fu tradotto dagli ingegneri in "Electronic Piece Of Cheese".

Più precisamente Symbian è il successore del sistema EPOC32. Fu creato nel 1998 grazie alla creazione della compagnia indipendente Symbian Limited, nata dall'aggregazione di numerose compagnie telefoniche e dalla stessa Psion.

Nel 2008 Nokia ha annunciato la volontà di rilevare le quote degli altri azionisti al fine di diventare l'unica proprietaria della piattaforma. L'intenzione è quella di renderla open source, mediante la creazione di Symbian Foundation; questa compagnia, formata dai vecchi azionisti e nuovi partecipanti, sarà dedita all'unificazione e la standardizzazione di tutte le interfacce, e all'introduzione della stessa agli sviluppatori esterni. L'obbiettivo è ultimare il lavoro e pubblicare il codice sorgente sotto la Eclipse Public License (EPL), licenza proprietaria di IBM che promuove lo sviluppo *open source,* entro il 2010.

CARATTERISTICHE PRINCIPALI DI SYMBIAN OS

La struttura Symbian è stata creata seguendo tre prerogative principali:

- importanza dell'integrità e della sicurezza dei dati utente
- salvaguardia del tempo dell'utente
- scarse risorse disponibili nel dispositivo

Per meglio adattarsi a queste esigenze è stato sviluppato con tali caratteristiche:

- **Microkernel EKA2** (EPOC Kernel Architecture 2); il *kernel* generalmente è un software avente il compito di controllare e gestire l'accesso all'hardware dei vari processi in esecuzione nel sistema operativo, assegnando ad ogni processo lo spazio in memoria, stabilendone un ordine di priorità, caricandone il codice in memoria ed eseguendolo. In particolare un *microkernel* offre un insieme ristretto di funzionalità, tra cui una gestione degli indirizzi a basso livello, l'amministrazione dei vari *thread* in esecuzione e un IPC (Inte - Process Communication) per la comunicazione fra i vari processi; si serve eventualmente di altre applicazioni per adempire ad altre richieste.

  L'EKA2 migliora la precedente versione 1 introducendo l'approccio *real-time* (attraverso cui le chiamate a libreria sono più veloci e di durata limitata) e utilizzando nuovi protocolli per l'interazione fra processi, compilatore e memoria.

  Con la tecnica real-time la CPU è abilitata a processare anche il *signalling stack* del telefono: nell'EKA1 tale insieme di protocolli per le comunicazioni di rete era gesito da una CPU dedicata.

  Tale *kernel* fornisce un importante vantaggio nella salvaguardia del tempo dell'utente e garantisce l'integrità e la salvaguardia dei dati utente. Inoltre permette ai dispositivi di contenere le proprie dimensioni, di essere più economici e di migliorare la propria efficienza in termini di consumi.

- La scelta di tale *kernel* consente di accedere ad altre prerogative quali **Multi-tasking** e **Multi-threading**; questi due concetti sono strettamente correlati fra di loro.

  Prima di introdurli chiariamo però la differenza fra "task" e "thread": un *task* è un insieme di istruzioni di un programma che vengono caricate in memoria; un *thread* invece è la suddivisione di un processo in più filoni concorrenti. Quando un programma viene compilato e tradotto in linguaggio macchina, si trasforma in una serie di istruzioni

che possono venire direttamente interpretate dal processore. Un *thread* non è altro che un insieme ridotto di queste istruzioni, indipendente dalle altre sequenze.

Possiamo quindi dire che il *multi-tasking* è la funzionalità attraverso la quale un sistema operativo gestisce più "compiti" contemporaneamente nello stesso processore; per *multi-threading* si intende invece un ulteriore livello di astrazione nella quale si suddivide un processo per ridurlo in elementi più facili da gestire, da eseguire comunque nello stesso tempo.

In particolare si parla di *pre-emptive multi-tasking* in Symbian. In un sistema mono processore, con tale strategia si da la possibilità allo *scheduler* (applicazione del sistema operativo che si occupa della gestione dei processi in esecuzione nella CPU) di interrompere momentaneamente un task e portarlo fuori dal contesto di esecuzione, per fare spazio ad un altro. Il task sostituto verrà poi ripreso in seguito. Tale azione viene denominata *context switch*. Così facendo si condivide il tempo di esecuzione della macchina, dando l'idea che i processi siano compiuti contemporaneamente, come in un sistema *multi-core*. In verità in Symbian solo ed esclusivamente un processo è eseguito ad ogni istante, essendo dedicato ad architetture *single core*, ma il *context switch* è svolto ad alte frequenze, dando l'idea della contemporaneità.

Il *multi - threading* è infine gestito tramite ISR (Interrupt Service Routine). Sempre per il fatto che il sistema è *single - core* vengono utilizzati due tipi di interruzione (Immediate Function Call, per rispondere immediatamente alla richiesta, e Deferred Function Call, risposta in differita) per compiere lo scambio dei *thread* in esecuzione.

- **Active Object**: è una forma di *multi - tasking* cooperativo attraverso la quale un task fa una chiamata a sistema; esso poi restituisce subito il controllo al processo, senza aver ultimato la sua richiesta. Il chiamante è libero di compiere altre azioni prima di tornare il controllo al sistema operativo. Quando la richiesta è completata quest'ultimo identifica il *thread* richiedente e lo riattiva, ripassandogli il controllo. Tale tecnica permette di risparmiare parecchie risorse del dispositivo.
- **Protezione della memoria**: è la capacità di controllare gli accessi alla memoria da parte di un processo, impedendogli ad esempio l'uso di uno spazio allocato per un altro task. È un servizio particolarmente importante per quanto riguarda l'integrità e la salvaguardia dei dati.

- È una struttura **basata su eventi**. Il sistema operativo risponde e determina il proprio comportamento in relazione agli eventi: tali eventi possono essere output di sensori, azioni da parte dell'utente, messaggi dai *thread*. Tale approccio è in contrasto con il *batch programming*, nella quale il flusso del programma è imposto dall'utente.
- Approccio **Request-and-Callback** ai servizi e separazione fra interfaccia utente e motore delle applicazioni. Per quanto riguarda il primo protocollo si tratta di una procedura con la quale un processo richiede un servizio ad una libreria attraverso una chiamata di sistema, ad esempio; per quanto concerne il secondo, si traduce nell'impossibilità da parte dell'utente di accedere ai servizi di basso livello del motore attraverso la UI (User Interface).

  Entrambe le caratteristiche permettono ancora una volta un'accelerazione nell'esecuzione dei processi.
- **Ottimizzazione per la salvaguardia dell'energia**: tutti i dispositivi che utilizzano tale sistema operativo montano batterie a bassa potenza, nella quale piccoli accorgimenti per la riduzione dei consumi sono fondamentali. Per raggiungere tale scopo si converte la CPU (Central Processing Unit) in modalità "basso consumo" quando non ci sono applicazioni in esecuzione e si gestisce la retroilluminazione del dispositivo, ad esempio, attraverso alcuni software; inoltre, si gestisce anche l'accensione e lo spegnimento dei vari sensori quali accelerometri o gps, i quali assorbono una non trascurabile percentuale dell'energia della batteria.
- **Sistema XIP e rientro in librerie condivise**. La prima caratteristica è l'acronimo di Execute In Place: con tale metodo si intende eseguire un programma direttamente dalla memoria di storage piuttosto che copiarla nella RAM. Per farlo ci si serve di alcuni accorgimenti, che sono: un'interfaccia simile tra CPU e disco di storage, o perlomeno un layer d'interfacciamento che sia sufficientemente rapido nelle operazioni di lettura/scrittura; un *file system* (qualora dovesse essere usato) che disponga di un'appropriata funziona di mappatura; programmi "linkati", affinché possano facilmente essere trovati in memoria; non devono inoltre modificare i dati caricati da quest'ultima.

  Per quanto riguarda il rientro in librerie condivise, si allude ad un direttiva secondo la quale una libreria utilizzata di recente può quasi sicuramente essere riutilizzata in breve tempo; è quindi necessario che i tempi di ri-entro in tale struttura siano rapidi.

Entrambe le prerogative permettono una riduzione dei tempi di esecuzione dei processi.

- **Organizzazione MVC** (Model-View-Controller): si tratta di un pattern architetturale per lo sviluppo di interfacce grafiche orientate ad oggetti, ossia un algoritmo di risoluzione di un problema ricorrente, in questo caso, appunto, la UI orientata ad oggetti.

  Si basa sulla divisione dei processi di un'applicazione fra Business Logic (applicativi che permettono l'interazione fra applicazione e utente), input per i processi e presentazione dei dati elaborati.

  Per implementare questa configurazione ci si avvale di tre entità, da cui poi ne scaturisce il nome: Model, cioè un insieme di processi che permettono l'elaborazione dati, View, che adatta i dati all'interazione con lo user, e il Controller, che si occupa degli input e li risolve con varie chiamate al "Model".

  Con tale paradigma è permesso scindere lo sviluppo di uno stesso software in tre fasi diverse, testando o correggendo ognuna separatamente. Tutto ciò consente un progresso nello sviluppo software per tale piattaforma.
- **Descrittori**: si occupano di gestire e conferire un comportamento prevedibile ai dati di tipo Stringa. In un qualsiasi sistema operativo, Symbian compreso, i dati di tipo Stringa possono rappresentare un problema per la loro amministrazione e per i loro comportamenti inaspettati. Evitano ad esempio problemi di *buffer overflows*, mettendo a disposizione una dozzina di classi per la manipolazione "sicura" di tali tipi di dati. Questo è un esempio determinante di protezione delle scarse risorse disponibili.
- **Cleanup Stack**: provvede a ripulire lo *stack* dagli oggetti per la quale i metodi utilizzati hanno generato eccezioni. Essenzialmente, creando un oggetto, gli si alloca uno spazio in memoria, appunto nello *stack* degli oggetti. Quando si chiama un metodo su di esso, possono essere generate delle eccezioni per un suo comportamento scorretto, che provocano l'interruzione dell'esecuzione. Se lo *stack* dovesse rimanere carico di oggetti "inutili", esso esaurirebbe il proprio spazio in poco tempo. Questo tipo di problematica è chiamata *memory leak*. Tale classe si avvale di semplici metodi per la risoluzione di questo problema, migliorando ancora una volta l'utilizzo delle scarse risorse di cui dispone il dispositivo. Si deve tenere conto che nella maggior parte dei casi la memoria è di tipo Flash; non se ne può quindi disporre in grande quantità visti i limiti di tale tecnologia.

- **Demand Paging**: è un applicazione di memoria virtuale. Permette di gestire meglio il caricamento dati dalla memoria, imponendo che una *disk page* venga caricata in memoria solo se esiste un effettivo tentativo di accedere ad essa. Il risultato è che l'avvio del processo è velocizzato ed inoltre si evitano svariati errori del tipo *page fault*.

STRUTTURA DEL SISTEMA OPERATIVO

La struttura di Symbian OS è una rappresentazione che spiega la composizione della struttura partendo dal più alto livello di astrazione per poi andare sempre più nel dettaglio, fino ad arrivare alle unità fondamentali. La configurazione è stratificata, gerarchica e dimostra delle dipendenze fra un'entità e l'altra.

Ci sono più rappresentazioni possibili della struttura, caratterizzate ognuna dal grado di dettaglio usato nella descrizione dei vari componenti. Si possono definire più livelli di astrazione:

- Layers
- Pacchetti
- Collezioni
- Componenti

Entrando nel dettaglio, i layers che compongono la struttura, dal basso all'alto livello, sono:

- **HAL**

  L'Hardware Adaption Layer è in grado di astrarre il comportamento hardware ai layers soprastanti, permettendo ad una qualsiasi applicazione di interfacciarsi con l'hardware del dispositivo senza preoccuparsi della sua precisa implementazione.

- **Core OS layer**

  Include il *Kernel Services* e gli *OS Services*. È una vera e propria piattaforma, indipendente da ogni componente attorno ad esso: indipendente dall'hardware al di sotto, e indipendente dalla GUI al di sopra.

- **Middleware layer**

  Rappresenta le funzionalità collegate alla User Interface e alle varie applicazioni. Si occupa di servizi vari alle applicazioni e altre competenze di alto livello, essendo indipendente dall'hardware grazie all'HAL.

- **Application layer**

  Contiene tutte le applicazioni disponibili per Symbian, nonché interfacce per il funzionamento delle stesse applicazioni.

Per quanto riguarda i pacchetti, essi sono raggruppati in Domini Tecnologici: all'interno di ognuno di essi i membri possono avere relazioni a livello di implementazione, o possono essere in rapporto per quanto riguarda l'evoluzione e lo sviluppo. Questi sono i Domini Tecnologici:

- Data Communications
- Device Connectivity
- Device Management
- Location
- Multimedia
- Multimedia Applications
- OS Base Services
- Personal Communications
- Productivity
- Runtimes
- Security
- Tools
- User Interface

In totale la struttura è costituita da 106 Pacchetti, 263 Collezioni e 495 Componenti.

PIATTAFORME PER SYMBIAN

Una piattaforma, in un sistema operativo, è essenzialmente la base su cui poter lanciare le applicazioni. Essa è rappresentata da un'architettura del processore, un linguaggio di programmazione, librerie di runtime oppure una GUI. Quando ad esempio dobbiamo sviluppare un'applicazione, potremmo teoricamente farlo direttamente in linguaggio *Assembly*; normalmente però, si preferisce utilizzare un linguaggio di alto livello, decisamente più *user – friendly*. Una piattaforma, in questo caso composta dal compilatore e dalle varie librerie annesse, permette di tradurre tali istruzioni in linguaggio macchina, consentendone l'esecuzione. Talvolta, in queste circostanze, si può ricorrere ad una *virtual machine*: una piattaforma simulata su un'altra piattaforma. Il suo compito è di simulare una base hardware e software per permettere ad una applicazione di essere correttamente eseguita.

Per quanto riguarda le piattaforme per Symbian OS, la più rilevante è senza dubbio la S60, essendo fra le più vendute al mondo. Tuttavia non è l'unica, in quanto esistono sul mercato anche la meno evoluta S40, la S80 dedicata ai Nokia Communicator e l'ultima nata Maemo 5.

Rilasciata e licenziata da Nokia, la S60 consiste in una serie di librerie e di applicazioni standard, quali ad esempio PIM (Personal Information Manager), players per audio e video di vari formati e la loro condivisione in rete, e browser web; le librerie invece comprendono pacchetti per lo sviluppo di applicazioni in vari linguaggi.

Difatti una delle caratteristiche che accomuna tutte le versioni della piattaforma S60 è il supporto nativo al linguaggio Java ME (attraverso la configurazione MIDP) e al Symbian C++. Per le versioni più recenti vengono introdotte anche librerie per Python e Adobe Flash.

Inoltre un'altra peculiarità che accosta queste varianti è la standardizzazione dei tasti: uno per l'accesso al menu, un joypad a 4 direzioni, due per la selezione e uno per la cancellazione dei dati digitati, inizio e termine chiamata, oltre che il tastierino numerico.

S60 è nata per offrire allo user una interfaccia più comprensibile e gradevole, facendola assomigliare ad un mini – personal computer: per raggiungere tale scopo supporta risoluzioni maggiori di 176 x 208 pixel, con profondità di colore a partire da 16 bit.

Il sistema operativo e la piattaforma utente sono collegati dalla seguente architettura:

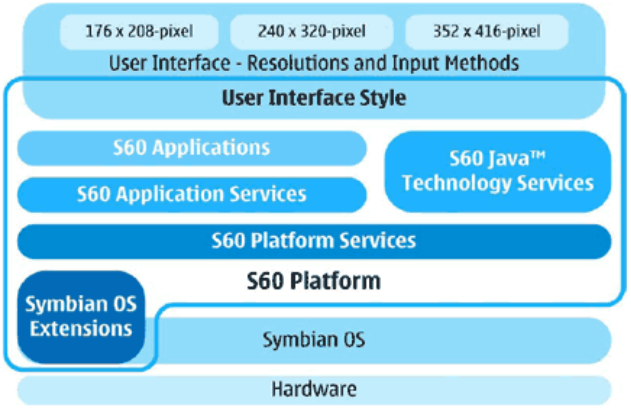

Fig. 1: architettura fra OS e User Interface

Si nota come al di sopra del Symbian OS, attraverso delle estensioni, viene costruita la piattaforma S60. Essa a sua volta si interfaccia con la UI, raggruppando all'interno vari layers per applicazioni e servizi, nonché un layer dedicato alla JVM (Java Virtual Machine).

La User Interface è l'entità attraverso la quale l'utente interagisce con il dispositivo: all'interno sono quindi raccolti i metodi per la gestione della risoluzione del display e degli input.

Esistono quattro diverse release dell'S60: serie S60 del 2001, serie S60 Second Edition del 2003, serie S60 Third Edition del 2005 e serie S60 Fifth Edition del 2008.

Come possiamo notare, manca la quarta edizione, in quanto il numero 4 venne scartato dalla casa madre secondo una politica aziendale per la quale tale numero rappresentava cattivi auspici in alcune credenze asiatiche.

La prima edizione, adottata dal Symbian OS 6.1, offriva l'implementazione del *WAP Protocol Stack* (un insieme di layers per la trasmissione di dati attraverso WAP), il supporto a Java MIDP 1.0, varie metodologie di comunicazione (sms, mms, infrarossi, bluetooth, seriale), connessione GPRS, librerie per i servizi multimediali, connettività e sincronizzazione con PC.

La seconda edizione, adoperata nei suoi diversi *Feature Pack* con Symbian 7.0s, 8.0a e 8.1a, aggiungeva alla vecchia piattaforma il supporto alla MIDP 2.0, il *browsing* su TCP/IP, la connessione EDGE, fotocamera avanzata, MP3 codecs, supporto a WCDMA, nuove risoluzioni e il supporto alla tastiera QWERTY.

La terza versione, utilizzata in due differenti Feature Pack con Symbian 9.1 e 9.2, introduceva l'EKA2, assieme a numerose librerie e un miglioramento delle funzioni di *browsing*.

Un utile strumento per la programmazione in questa piattaforma è l'emulatore. Si tratta di un software in grado di testare un applicazione rivolta a Symbian OS in un sistema operativo differente, ad esempio in uno dedicato a PC. Compreso nella SDK (Software Development Kit), si presenta all'utente come una copia fedele del proprio dispositivo, riproducendo precisamente il suo comportamento, anche dal punto di vista grafico. Qualora si dovesse creare un'applicazione, sarebbe favorevole testare il suo comportamento prima dell'installazione: attraverso l'emulatore possiamo provvedere a tale esigenza, determinando se il programma svolge le sue attività senza errori inattesi.

Symbian S60 Fifth Edition

La più recente versione dell'S60 è la quinta edizione. Essendo un pacchetto software, essa risulta più ampia rispetto al sistema operativo sottostante, il Symbian 9.4. La novità più rilevante è l'introduzione della Touch UI, abbinata al feedback tattile: essa rappresenta un nuovo tipo di interazione con l'utente, consentendo l'uso di applicazioni che supportano l'uso di keypad e touchscreen contemporaneamente.

Durante l'implementazione di un programma, lo sviluppatore deve accertarsi che esso non dipenda da una tastiera o keypad, ma che sia abilitato all'uso dello schermo touch. Il feedback tattile è realizzato invece facendo vibrare leggermente il dispositivo quando un comando è toccato sullo schermo. I metodi che agiscono sui componenti Touch UI sono regolati dalle librerie AVKON.

Quando lo schermo viene toccato, l'applicazione in *foreground* riceve due tipi di indicazione: *pointer up* e *pointer down*. A seconda dell'applicazione stessa, il comando viene interpretato per cambiare vista oppure per selezionare un oggetto. Se si vogliono interpretare altri tipi di interazione, quali possono essere il trascinamento, lo sfioramento o il tocco prolungato, devono essere implementate altre API oltre le standard. Questi movimenti possono essere utili nella manipolazione di file, oppure nello scorrimento di immagini. La particolarità dell'implementazione hardware dello schermo, di tipo resistivo, mal si adatta all'interpretazione del movimento di trascinamento; quest'ultimo viene spesso confuso con l'evento *pointer up*, creando disagi all'utente. D'altro canto tale tecnologia permette una riduzione dei costi, nonché una durata maggiore del prodotto.

L'interfaccia utente è concepita usando svariati componenti; la ricerca adattativa sviluppata in tale piattaforma si abbina al *Find Pane*: per trovare un nome in rubrica, ad esempio, viene presentata una tastiera in cui sono mostrate solo le lettere che effettivamente compongono un nome presente in essa. La ricerca quindi si "adatta" ai risultati possibili.

Gli *AVKON Buttons* sono invece i componenti base del sistema e possono venire utilizzati anche in oggetti più sofisticati; essi possono essere arricchiti da un testo o un immagine, e un loro tocco corrisponde ad una determinata azione.

La *toolbar*, invece è un insieme di *Buttons*, la quale può essere di dimensione fissata (al massimo tre pulsanti) o *floating* (numero variante di componenti).

Esistono poi dei *Pop – Up Menu*, in grado di raccogliere numerosi oggetti e di presentarli all'utente. I *Timers* sono spesso utilizzati per temporizzare tali menu: impostando un valore di tempo, il menu può sparire dopo un determinato numero di secondi o rimanere in *foreground* finché lo user non interviene.

Infine, la *Choice List* rappresenta una lista all'interno della quale un singolo oggetto può venire selezionato.

I metodi per l'inserimento dei testi sono standardizzati secondo l'ITU-T (Internation Telecommunication Union - Telecommunication Standardization Bureau): nel momento in cui un editor viene abilitato, è possibile immettere dati utilizzando una tastiera virtuale oppure il riconoscimento della scrittura a mano. Entrambi i metodi regolano la propria orientazione (verticale o orizzontale) a seconda dell'orientamento del dispositivo. Per raggiungere tale scopo, nonché per svolgere altre funzioni, il dispositivo si avvale dei dati raccolti dai nuovi sensori introdotti: accelerometro, magnetometro e sensori di tocco. Ognuno di essi è regolato da *framework* appositi, in grado di interpretare ed utilizzare i dati provenienti da tali apparati. Infine per quanto riguarda lo schermo, la risoluzione passa a 640 x 360 pixels, prendendo il nome di nHD.

In S60 Fifth Edition vengono presentati gli *widget*, oltre che la possibilità di sviluppare software su base Flash Lite e il supporto agli Windows Media audio e video file, permettendo la stessa cifratura di tali files sia nel PC che nel dispositivo mobile. Il *widget* non è altro che un elemento grafico, appartenente alla UI che permette una più rapida interazione con una determinata applicazione, rappresentando le sue informazioni primarie direttamente sulla schermata principale del sistema. Tale interazione è permessa dal WRT (Web Runtime) il quale a tal scopo estende le funzionalità del Web Browser presente nel sistema operativo.

Per quanto riguarda il management delle applicazioni installate, ci si riferisce alla piattaforma UID: ogni software dedicato all'S60 viene contraddistinto da una stringa a 32 bit. Se si tenta di installare un'applicazione per S60 Third Edition in tale piattaforma, appare un messaggio di errore che notifica tale evento.

Il nuovo emulatore adottato per questa edizione supporta la risoluzione nHD. Le funzioni di debugging sono rese più efficienti, velocizzando i tempi di caricamento dell'emulatore stesso; per farlo vengono limitati i servizi caricati immediatamente, rendendoli disponibili in futuro se dovessero rendersi necessari: tale meccanismo è regolato da due "modi d'uso" settabili: *Limited Services Mode* e *All Services Mode*.

LINGUAGGI PER SVILUPPO SOFTWARE IN SYMBIAN S60 FIFTH EDITION

Esistono svariati linguaggi per l'implementazione di software indirizzato al Symbian S60 Fifth Edition. Saranno descritti brevemente e verrà elencato il materiale necessario allo sviluppo delle applicazioni. Particolare attenzione sarà dedicata invece al linguaggio Java, alla quale è dedicato un capitolo a parte e con la quale verrà infine realizzato un applicativo stile "Hello World" destinato all'istallazione su Nokia N97, caratterizzato appunto dal disporre di tale sistema operativo e interfaccia utente.

Symbian C++

È il linguaggio nativo del sistema operativo Symbian, pur non rappresentando lo standard d'implementazione. Oltre alle librerie tipiche di C++, supporta anche lo standard Open C (software tool per l'analisi e il debug di applicazioni in C++), senza il bisogno di installare estensioni alla piattaforma grazie alle API dedicate.
È previsto anche l'accesso a POSIX (Portable Operating System Interface for UniX), la famiglia di standard unificati dalle IEEE per la definizione di API per software compatibili con le varie versioni di Unix OS, oltre che l'introduzione di svariate librerie indirizzate a vari ambiti di programmazione.
Per realizzare un'applicazione per Symbian S60 Fifth Edition in tale linguaggio sono necessari alcuni prerequisiti, da installare nel proprio PC:

- SDK (Software Development Kit), utile strumento per lo sviluppo e il test dell'applicazione in C++. Oltre a una svariata collezione di API, librerie, documentazione ed esempi, prevede anche un GCCE (Gnu C Compiler Embedded), un compilatore dedicato alla costruzione di programmi per dispositivi mobili.
- IDE (Intregrated Development Enviroment) è letteralmente un ambiente di sviluppo integrato, costituito da una famiglia di software, destinato a facilitare la creazione dell'applicazione. Un esempio ne è Carbide C++, sistema basato su Eclipse che permette di utilizzare un'interfaccia semplice e *user-friendly* che segua il programmatore in ogni fase dello sviluppo.
- Compilatore, un programma dedicato generalmente alla traduzione di un programma sorgente, scritto in codice di alto livello, in un linguaggio binario.

A seconda dell'obiettivo, ne esistono tre tipi: WINSCW, per programmi dedicati al test su emulatore, è compreso nell'IDE Carbide C++; GCCE, per programmi dedicati all'installazione su dispositivi mobili, è parte integrante dell'S60 SDK; ARMV5, anche quest'ultimo dedicato ad applicazioni per apparati mobili, permette di incrementare le performance del processore ARM ed è reperibile separatamente nel sito del costruttore.

- Command line tool, configurazioni attraverso la quale è possibile fare a meno dell'IDE e seguire lo sviluppo dell'applicazione direttamente da riga di comando, eseguendo particolari procedure in background.

Il risultato della creazione di un'applicazione definita in Symbian C++ è un pacchetto di files, caratterizzati dall'estensione .sis, destinati ad essere installati nel dispositivo.

Qt

Qt è una libreria multipiattaforma che isola l'utente il più possibile dalle differenze dei vari sistemi operativi. Servendosi di Qt, l'utente può costruire un'applicazione una volta per tutte, utilizzandola poi su diversi desktop e sistemi operativi. Lo standard seguito da tale linguaggio deriva dal C++: ciò significa che per l'utente che già conosce tale sintassi è sufficiente approfondire pochi dettagli per poter sfruttare questa potente multipiattaforma.
È possibile realizzare anche applicativi con interfaccia grafica, utilizzando gli *widget* caratteristici di tale linguaggio, che sono simili ma non identici a quelli adottati dalla S60 UI.
Gli strumenti di sviluppo sono esattamente gli stessi del linguaggio nativo: SDK, Carbide C++ (facendo attenzione alla versione e alle impostazioni), emulatore e compilatore; in aggiunta si installa la distribuzione Qt, in versione compatibile con Symbian OS.

Web Runtime Environment (WRT)

I dispositivi basati su Symbian S60 Fifth Edition sono in grado di supportare i seguenti contenuti Web, scritti in svariati linguaggi:

- Pagine Web da aprire attraverso un browser installato nel sistema operativo
- *Widgets*, cioè applicazioni che usufruiscono dei contenuti Web, ma che sono direttamente installate nel sistema operativo e che quindi offrono un accesso alla rete senza l'utilizzo di nessun browser

Per quanto riguarda lo sviluppo di pagine Web, le conoscenze non sono diverse da quelle richieste per lo sviluppo delle stesse in ambiente dedicato a PC.

La differenza sostanziale che esiste fra un browser di un PC e quello di un dispositivo mobile è la dimensione (e conseguentemente la risoluzione) dello schermo, oltre che le limitate risorse di cui è in possesso quest'ultimo. Per aggirare tale ostacolo, spesso vengono create pagine Web ottimizzate per tali dispositivi; per farlo può essere creato un URL ad - hoc, oppure con una soluzione lato server si provvede allo scaricamento parziale dei contenuti, in risposta una connessione da parte di tali apparecchi.

Per lo sviluppo di pagine Web sono richiesti alcuni componenti tra i quali: un PC per l'accesso alla rete, un *text editor* o un IDE per lo sviluppo dei files richiesti dal sito, programmi grafici qualora si volessero utilizzare immagini, un browser per PC ed infine un potente quanto utile strumento per testare la propria creazione, come il *Remote Device Access*. Quest'ultimo, messo a disposizione da *Forum Nokia*, permette di selezionare un qualsiasi tipo di dispositivo mobile, ed attraverso un applicativo Java, ne viene simulato fedelmente in remoto il comportamento. È quindi possibile selezionare il browser Web integrato e digitare l'URL che si vuole verificare, per farlo comparire sullo schermo del proprio PC. Altri metodi per il test delle pagine create sono il trasferimento delle stesse nel dispositivo, il che permette di accedervi senza la connessione, oppure l'uso dell'emulatore incluso nel SDK.

Per quanto riguarda invece gli *widgets*, essi possono venire installati nel dispositivo mobile attraverso files con estensione .wgz. Quest'ultimi sono generati in ambiente WRT (Web Runtime) e necessitano dei medesimi strumenti elencati in precedenza per poter essere sviluppati. Permettono di interagire con l'utente attraverso l'immissione di input, possono aggiornarsi automaticamente, possono gestire il lancio di applicazioni di cui essi necessitano e possono ispezionare un contenuto Web.

Particolari *widgets*, detti *Home Screen Widgets*, possono essere rappresentati nel desktop del dispositivo mobile, permettendo all'utente di visualizzare i dati ad esso relativi senza dover aprire a pieno schermo l'applicazione. Non sono interattivi, ma un loro semplice click consente di accedere al *widget* (che finora era in esecuzione in background); inoltre l'interazione fra *Home Screen* e *widget* avviene a livello di sistema.

Flash lite

Adobe Flash è un *authoring tool*, cioè un pacchetto software che l'utente utilizza per creare dei contenuti in un determinato ambito, dedicato allo sviluppo di applicazioni multimediali interattive per Internet e altri ambienti. Flash Lite è un sottoinsieme delle funzionalità offerte dall'Adobe Flash Player, il programma dedicato alla presentazione di tali formati. Esso consente allo sviluppatore di creare contenuti multimediali interattivi in ambiente mobile, utilizzando il linguaggio caratteristico di tale piattaforma: *Adobe ActionScript*.

Le potenzialità di tale tecnologia sono:

- Animazioni e giochi interattivi
- Riproduzione di contenuti multimediali
- Interfaccia utente avanzata e facilità d'uso
- Sviluppo rapido

Per lo sviluppo di applicazioni in Flash Lite ci si serve anche di un'applicazione di test, chiamata *Adobe Device Central*, che assiste l'utente permettendo di collaudare la propria creazione su vari devices. Nel realizzarla, ci sono significative linee guida alle quali attenersi, per dare origine ad un buon software; ad esempio, le differenti capacità della propria apparecchiatura, in relazione alla risoluzione e dimensione dello schermo piuttosto che alla disponibilità di memoria. Tali considerazioni distinguono l'ambiente Flash dedicato a PC da quello dedicato a dispositivi mobili, lasciando la maggior parte delle ulteriori caratteristiche inalterate: ciò significa che la migrazione di un applicativo da PC a telefono non genera gravi complicanze.

Le applicazioni create in Adobe Flash Lite possono essere distinte in tre categorie:

- Stand – alone, applicazione con estensione .swf installata direttamente nel dispositivo e eseguita dal player opportuno
- Contenuti integrati per browser, inclusi in pagine HTML o XHTML, scaricati assieme alla pagina Web e presentata attraverso una plug – in adottata dal browser
- *Screen saver*, è un file .swf che non richiede l'intervento dell'utente per essere eseguito. Uno *screen saver* è attivato dopo un tempo settato di inutilizzo del dispositivo, ed è in primo piano finché la retroilluminazione è attiva

Per lo sviluppo di applicativi per Flash Lite, oltre al PC, sono necessari:

- Adobe Flash *authoring tool*, scaricabile dal sito del produttore

- *Adobe Device Central*, un software per il test del programma, già introdotto precedentemente
- Un dispositivo mobile adatto per l'installazione dell'applicazione; diversi telefoni supportano diverse versioni di Flash Lite, caratterizzate da differenti potenzialità.

Python

Python è un linguaggio ad alto livello, che supporta diversi paradigmi di programmazione (orientato ad oggetti, imperativo e funzionale) ed offre una tipizzazione dinamica forte; ciò significa che supporta svariati tipi di dati, permettendo che una variabile dichiarata di un certo tipo all'inizio del programma, sia convertita in un altro durante l'esecuzione.
Dispone di una libreria molto ricca, nonché di robusti costrutti per la gestione delle eccezioni e un meccanismi di *garbage collector* per l'amministrazione della memoria. Tutte queste caratteristiche lo rendono facile da usare, nonché intuibile e di rapido apprendimento.
Inoltre è un linguaggio pseudo – compilato, cioè un interprete si incarica del controllo sintattico del listato e, se questo dovesse risultare positivo, si procede all'esecuzione. Non esiste quindi un file eseguibile generato dal codice sorgente. Questa peculiarità lo rende portabile, ossia una volta scritto il programma, è possibile interpretarlo sulla quasi totalità delle piattaforme disponibile, attraverso l'uso dell'interprete adatto.
S60 supporta Phyton, offrendo un alto grado di integrazione con sistema operativo e interfaccia grafica, facendo sì che le applicazioni realizzate con tale sintassi siano molti simili a quelle native del dispositivo stesso.
Per provvedere allo sviluppo in Python dobbiamo disporre di:

- L'ambiente runtime di Python, da installare nel dispositivo mobile
- *Pyhton Script Shell*, che abilita l'esecuzione delle applicazioni Python nel dispositivo
- S60 SDK per Python, il quale completa i due precedenti strumenti fornendo un tool software in grado di seguire lo sviluppo e il test dell'applicazione

Un altro utile strumento, non fondamentale peraltro, è la Python consolle; da installare nel dispositivo, permette lo sviluppo del programma direttamente on device. Inoltre, attraverso il supporto alla connessione Bluetooth, è possibile seguire tale processo dal proprio PC, attraverso un emulatore. È concesso in ogni caso stilare il listato attraverso l'uso di un IDE per Python, anch'esso installato nel proprio PC.

JAVA ME SU SYMBIAN S60 FIFTH EDITION

Java ME (Micro Edition) è la versione ottimizzata della piattaforma Java dedicata ai dispositivi mobili. Rappresenta l'ambiente di *runtime* più fruibile per questa tecnologia, essendo largamente disponibile in molti devices.

Le novità introdotte rispetto alle edizioni precedenti riguardano soprattutto la Touch UI (che come spiegato in precedenza, permette all'utente di interagire con il dispositivo attraverso lo schermo *touch screen*), le relative modalità di introduzione del testo, e la risoluzione nHD.

La piattaforma Java è disponibile in varie versioni, ognuna dedicata a varie architetture caratterizzate da diverse capacità. Oltre alla Micro Edition, sono disponibili la Standard Edition e la Enterprise Edition, dedicate rispettivamente a workstation e server.

Le tre edizioni sono simili nella loro base e nelle caratteristiche generali: sono presenti infatti funzioni di *garbage collection* (liberazione della memoria erroneamente occupata da oggetti dereferenziati) e indipendenza dal sistema operativo utilizzato grazie alla *virtual machine*. Quest'ultima proprietà permette ai programmi costruiti in Java di poter essere eseguiti su qualunque base o architettura, dando senso alla massima "*Write once, run everywhere*"; infatti una macchina virtuale è un'implementazione software di un dispositivo (come un computer) che esegue l'applicativo esattamente come una macchina fisica. Per farlo, la JVM (Java Virtual Machine), si serve di una speciale forma di linguaggio intermedio detto Java Bytecode: lo stesso programma scritto in una determinata architettura, compilerà poi ovunque, sfruttando appunto la traduzione in tale codice. Pur essendo simile a quello delle altre edizioni, il Java Bytecode per Java ME occupa meno spazio in memoria, necessitando però di un controllo della sintassi prima della compilazione.

A differenza delle altre edizioni, invece, Java ME è dedicato a dispositivi integrati, i quali dispongono generalmente di memoria limitata e scarse risorse; tuttavia condivide svariati pacchetti con le edizioni simili, permettendo un facile adattamento del programma da un'architettura ad un'altra.

Le librerie per questa piattaforma sono divise in due insiemi, dette configurazioni: Connected Device Configuration (CDC) e la più ristretta Connected Limited Device Configuration (CLDC).

Le configurazioni si dividono ulteriormente in profili, i quali si concentrano ancor di più su specifici dispositivi. Ad esempio, il MIDP (Mobile Information Device Profile) aggiunge librerie

dedicate a dispositivi mobili o PDA (Personal Digital Assistant, comunemente detti palmari). Un applicativo dedicato a tale profilo è eseguibile in un qualsiasi device provvisto di tali librerie.
Le librerie messe a disposizione da tale profilo forniscono una base per i requisiti del programmatore.

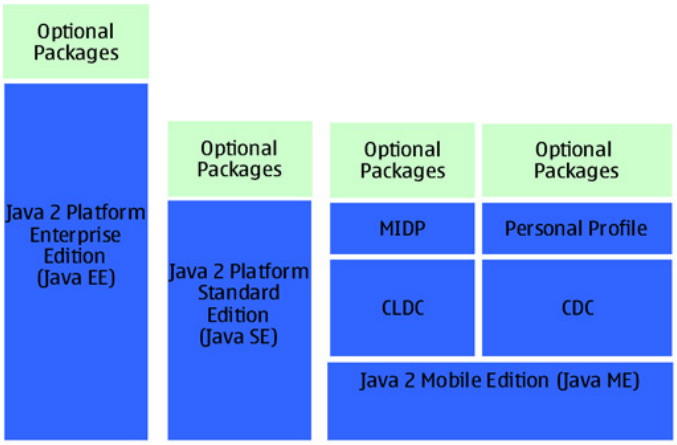

Fig. 2: edizioni Java con librerie annesse

In aggiunta a queste vengono messe a disposizione librerie più dettagliate, riguardanti ambiti più specifici quali disegno di vettori grafici e realizzazione di crittografia avanzata.
L'implementazione di tale profilo è conforme al MSA (Mobile Service Architecture), detta anche JSR – 248. Una JSR (Java Specification Request) è un documento formale con la quale si propongono nuove tecnologie o specifiche da aggiungere alla piattaforma. In questo caso la MSA è una specifica che descrive un ambiente wireless end – to – end per Java, che comprende altre 16 JSR e 8 sottoinsiemi di JSR.
La conformità a questa architettura rende le API di Java meno frammentate, fornendo un utile strumento per uniformarne l'implementazione.
Svariate versioni della Java ME sono disponibili; per poter implementare un qualsiasi software è fondamentale essere a conoscenza della versione in uso nel dispositivo. Inoltre è possibile aggiornare l'ambiente di *runtime*, attraverso l'update del telefono. Ciò significa che non è possibile legare un determinato modello di cellulare alla propria piattaforma Java, poiché quest'ultima può essere diversa in due modelli identici, grazie ai suddetti aggiornamenti.
Le applicazioni generate in Java con il supporto del profilo MIDP sono dette MIDlets Applications. Esse sono eseguibili su dispositivi compatibili e forniti delle librerie necessarie.
Per essere classificata come MIDlet un'applicazione deve soddisfare determinati requisiti:

- Estendere la classe astratta javax.microedition.midlet.MIDlet, che ne controlla il ciclo di vita
- Essere raccolta in un pacchetto JAR (Java ARchive)
- Contenere all'interno del file JAR un file MANIFEST.MF
- Disporre di un file JAD (Java Application Descriptor)
- Aver verificato sintatticamente i file .class, azione svolta dai tool di sviluppo

Un file JAR è un archivio compresso che può essere gestito con la maggior parte di programmi dedicati a questo tipo di dati, compreso Win Zip. Contiene più tipi di file, quali i compilati .class, altri file di risorse multimediali e il file MANIFEST.MF, avente funzioni simili al JAD. Quest'ultimo è usato anche per definire contenuto e gestione della MIDlet. È un file imprescindibile dall'archivio, ogni file JAR è necessariamente accompagnato da un JAD.

Più file MIDlet possono essere contenuti all'interno dello stesso file JAR. In questo caso la raccolta prende il nome di MIDlet Suite, e ognuna delle applicazioni è installata singolarmente nel device.

All'atto del runtime, godendo il Symbian OS del *Multi – Tasking*, è possibile eseguirli contemporaneamente nella stessa JVM.

Inoltre, all'interno della stella JVM, vista dal Symbian come un normale processo, è possibile far girare più MIDlet Suites; nonostante tutto non è possibile richiamare una MIDlet da un'altra MIDlet, come non è possibile farlo per la MIDlet Suite. In più, nessuna risorsa può essere condivisa fra MIDlet appartenenti a due diverse suites, se non con particolari permessi.

In ogni caso la condivisione di risorse fra MIDlet è permessa dall'RMS (Record Managment System) ossia uno spazio allocato in memoria (la stessa dove è installata l'applicazione) diviso in campi di bit di lunghezza arbitraria, nella quale sono raccolte le informazioni da condividere, contraddistinte da nomi diversi.

In sintesi, in Java è permesso il *Multi – Threading* per applicazioni appartenenti a medesime suite, ma è precluso il *Multi – Tasking* per quelle appartenenti a suite diverse e non è supportato l'uso di gruppi di thread, cioè thread raggruppati e gestiti insieme, oltreché l'uso di thread daemon, processi speciali utili ad amministrare la fine dell'esecuzione del programma (entrambi regolati correttamente nelle altre versioni Java SE e Java EE).

Per implementare una MIDlet è necessario disporre di alcuni tools specifici, quali:

- Java SE Software Development Kit, richiesto per compilare il Java Bytecode sul PC utilizzato per lo sviluppo dell'applicazione

- IDE, che come già detto, è letteralmente l'ambiente integrato per lo sviluppo dell'applicazione
- SDK che supporti la Java ME e che disponga dell'emulatore per il dispositivo nella quale verrà installato l'applicativo
- Un dispositivo mobile per il test dell'applicazione, assieme ad un mezzo valido per la comunicazione con un PC; solitamente i collegamenti più comuni sono USB o Bluetooth.

Esistono principalmente due IDE per Java ME: Eclipse, distribuito da Eclipse Foundation, dispone del MTJ (Mobile Tools for Java), consistente in un plugin per il supporto di Java ME.

NetBeans invece, è scaricabile in varie versioni, distinte dal linguaggio di programmazione supportato; non servono plugin aggiuntivi, in quanto la versione scaricata per una determinata piattaforma è già completa.

Come primo step utile per poter iniziare lo sviluppo è necessario aggiungere un SDK all'IDE; dopo aver installato entrambi i tools, si provvede a settare le impostazioni dell'ambiente di sviluppo affinchè riconosca l'SDK come piattaforma Java disponibile.

Una volta installati e settati tutti i software necessari si può provvedere all'analisi del progetto: devono essere esplicitati requisiti, scopo e funzionalità, nonché metodo per l'installazione del programma.

Si analizza poi l'ambiente per lo sviluppo, in accordo con le caratteristiche degli IDE disponibili.

Si crea successivamente il codice sorgente in linguaggio ad alto livello, attraverso un qualsiasi *text editor*, che darà origine all'applicazione; esso sarà caratterizzato dall'estensione .java.

Una volta ultimato, si passa alla compilazione delle classi, non prima di aver svolto la preverifica della sintassi, compito svolto direttamente dall'IDE.

Il risultato è un file .class, in linguaggio Java Bytecode, direttamente interpretabile dalla JVM; dopo di che è possibile creare il pacchetto JAR ed il file JAD (operazione svolta automaticamente dall'IDE nel processo di costruzione/installazione), che verranno poi testati con l'emulatore messo a disposizione dall'SDK.

L'IDE mette anche a disposizione un editor per il file JAD, in grado di settare direttamente le caratteristiche disponibili. Se la verifica dovesse andare a buon fine, attraverso il metodo prescelto si installa la MIDlet creata nel dispositivo mobile, provvedendo all'eventuale distribuzione sul mercato.

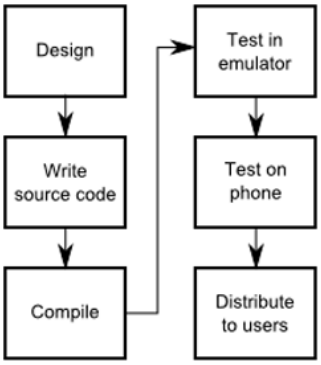

Fig. 4: routine di sviluppo applicazione

Il progetto ultimato, dal punto di vista dei files creati, è gestito alla stessa maniera indipendentemente dall'IDE scelto. I files sono divisi in cartelle:

- src, la cartella contenente il file sorgente .java del programma
- bin o build, cartella nella quale sono raccolti i files .class, preverificati e compilati dai file sorgente
- res, cartella che raccoglie file multimediali opzionali che possono essere supportati dalla MIDlet
- deployed o dist, cartella contenente gli archivi JAR generati per l'installazione su dispositivo mobile

Per quanto riguarda il ciclo di vita della MIDlet, esso è controllato dall'AMS (Application Management Software). Il ciclo di vita è inteso come il tempo che intercorre fra l'installazione e la disintallazione dell'applicazione dal dispositivo.

Il fulcro di questo periodo è rappresentato però dall'esecuzione della stessa: quando l'utente seleziona un applicativo, l'AMS lo istanzia richiamandone il metodo startApp(), il quale dà inizio al ciclo di esecuzione.

Tale software di gestione controlla ciò che è mostrato sul display, distinguendo fra l'applicazione in *foreground* e quelle in *background*: la prima è unica ed è mostrata all'utente, le seconde possono essere multiple (non c'è limite formale al numero, se non la memoria disponibile) e sono in esecuzione senza che il display ne dia visione.

Una volta iniziata l'esecuzione, il controllo è ceduto al programma stesso, che lo cederà nuovamente all'AMS nel caso in cui debba essere passato in *background* o debba essere terminato.

L'AMS amministra inoltre le funzioni che hanno bisogno di un'elaborazione immediata, quali sms o chiamate in entrata; quando una di queste funzioni dovesse sopraggiungere, l'applicazione in *foreground* viene relegata in *background* per fare spazio all'elaborazione dell'applicazione a priorità maggiore.

Le applicazioni sopra citate (sms, chiamate entranti, allarmi etc.) sono servizi disponibili in ogni momento in *background* e sono raccolte in un apposito spazio detto *Push Registry*; quando un evento ne richiede l'attivazione, l'AMS interviene invocando il metodo startApp() su di esse, causando l'interruzione dell'esecuzione di una qualsiasi applicazione in *foreground*.

Per essere compatibile al trattamento con AMS una MIDlet deve però implementare alcuni metodi:

- startApp(): già sopra citato, è il metodo per l'inizio dell'esecuzione. Quando è invocato, il sistema carica in memoria tutte le risorse occorrenti; un'eccezione generata in questo momento causerebbe l'immediata terminazione dell'esecuzione
- pauseApp(): è un metodo che normalmente non è utilizzato e solitamente risulta vuoto, ma dev'essere obbligatoriamente implementato; può essere abilitato e usato mediante un attributo nel file JAD
- destroyApp(): determina la fine dell'esecuzione; vengono liberate le risorse occupate e vengono distrutti gli oggetti generati. L'applicazione dispone di 5 secondi per adempire alla chiamata di tale metodo, dopo di che interviene l'AMS.

  La MIDlet può essere chiusa in vari modi:

  - Pulsante di controllo "End"
  - Dall'AMS
  - Rimozione della memory card
  - Esaurimento della memoria disponibile

Prima di essere definitivamente installata nel dispositivo mobile, l'applicazione necessita di svariati test, atti a verificarne il corretto comportamento.

Per raggiungere tale scopo, che per sicurezza non può essere subito svolto nel dispositivo, l'SDK mette a disposizione un emulatore. Quest'ultimo offre un'imitazione fedele del sistema operativo montato dalla periferica, nel quale l'elaborazione e lo *scheduling* delle applicazioni è pressoché identico a quello preso a modello. Con tale tool è possibile anche simulare degli eventi per verificare la risposta del dispositivo; questi possono essere ad esempio la rimozione

della memory card, ricezione di un sms, disconnessione dalla rete e l'allocazione di un disco usando lo spazio di memoria del PC.

È inoltre possibile utilizzare la finestra di diagnosi per rilevare diversi tipi di dati, quali il traffico HTTP del browser, i messaggi provenienti dallo standard System.out delle applicazioni e la lista delle applicazioni in esecuzione, complete di percentuale di utilizzo della CPU e della memoria, in stile Task Manager.

Un'altra via per la diagnosi e risoluzione dei problemi è messa a disposizione dal *framework* ECMT, che supporta la EcmtAgent; questa applicazione installata nel dispositivo mobile, necessita di dialogare con la *Device Connectivity Tool*, anch'essa messa a disposizione dall'SDK ed installata nel PC. L'applicazione lato dispositivo mobile serve a stare in ascolto delle connessioni in arrivo, ed è necessaria ogni qual volta si voglia stabilire una connessione fra PC e dispositivo a scopo di test. La connessione può essere basata su USB, Bluetooth e WLAN. L'applicazione lato PC è usata anch'essa per instaurare tale connessione, monitorando il comportamento del programma testato.

Con l'ODD (On – Device Debugging) è possibile compiere il debug dell'applicazione direttamente sul dispositivo interessato, disponendo di strumenti simili a quelli messi a disposizione dall'IDE.

Il passo finale, dopo il test dell'applicazione, è la sua installazione.

Esistono svariati metodi a tal proposito:

- OTA (Over – The – Air), metodo il quale si avvale del download da WEB dell'applicativo
- Connessione diretta con PC, su protocollo USB, Bluetooth o Ir, necessita di un tool adatto come il Nokia PC Suite
- E – mail e MMS, mediante l'uso di allegati possono essere spediti i file JAR e JAD, che appena vengono selezionati causano l'attivazione dell'AMS
- Pre installazione della MIDlet su dispositivi removibili come memory card. Nel caso in cui l'installazione dovesse andare a buon fine l'applicazione sarebbe spostata nel disco locale dell'apparecchio, oppure potrebbe essere lasciata nella memoria removibile e essere eseguita correttamente.

Per quanto riguarda invece l'installazione di MIDlets già esistenti attraverso OTA, messe online da un qualsiasi sviluppatore, si segue una procedura ben precisa: attraverso il browser web del dispositivo si raggiunge il link per l'installazione dell'applicazione; una volta richiesta la MIDlet il server risponde inviato il file JAD, con la descrizione del contenuto dell'archivio JAR. Dopo aver

confermato l’installazione, il server provvede a inviare il file JAR, il quale verrà scompattato e installato.

IMPLEMENTAZIONE APPLICAZIONE "HELLO WORLD"

Il codice d'esempio è un'applicazione in stile "Hello World". Ciò significa che al suo lancio, ci si aspetta che compaia una finestra di testo, con una stringa di testo contenente la frase "Hello World".

Per implementarla, ci serviamo dei seguenti tool:

- Java SE Development Kit 6 Update 13
- NetBeans IDE 6.7.1
- Nokia N97 SDK v1.0
- Nokia Ovi Suite 2.0.1.35
- Nokia N97 tipo RM – 505 versione software V 11.0.021 su Symbian OS v9.4 S60 5th Ed.

Il programma consiste nella stesura di due files: HelloWorldMIDlet.java e HelloScreen.java: il primo rappresenta il punto di accesso dell'intera applicazione, il quale genera il testo che sarà rappresentato nella finestra di dialogo, creata dal secondo file.

Il flusso di progetto per creare la suddetta applicazione è il seguente:

- Creazione di un nuovo progetto MIDP
- Creazione del codice sorgente
- Costruzione del progetto
- Installazione nel dispositivo

Creando un nuovo progetto in NetBeans, vengono offerte diverse possibilità; per i nostri scopi, utilizziamo la piattaforma Java ME e generiamo un'applicazione MIDP.

Dopo aver settato nome e cartella di destinazione, viene proposta l'opzione "Create Hello MIDlet": spuntandola, l'IDE abilita il suo *visual designer*, il quale attraverso il *flow design* abilita la costruzione di finestre per l'applicazione mediante il trascinamento di oggetti o pulsanti, le cui proprietà sono settate in finestre apposite; in alcune occasione diventa molto utile, ma per tale progetto non è favorevole.

Si passa poi alla scelta dell'emulatore, nel qual caso scegliamo quello messo a disposizione dal nostro SDK; se stiamo creando la versione finale del progetto, dedicata al dispositivo mobile e non più all'emulatore, lo si può definire attraverso l'apposita opzione.

La creazione del listato riguarderà un paragrafo a parte a fine capitolo, corredato da un commento ai metodi utilizzati nel programma.

Una volta generati i file .java, può essere costruito il progetto; con il comando “Build Project” il programma è compilato e, salvo errori di sintassi, vengono generati i file .class. Si può procedere quindi al test sull’emulatore usando il comando “Run Project”.

Dopo una serie di azioni, l’emulatore viene aperto e si può verificare direttamente il comportamento dell’applicazione. Si precisa che nella S60 5th edition, poiché l’interfaccia è la Touch UI, il tocco dello schermo del dispositivo corrisponde ad un click del mouse nell’emulatore.

Per procedere all’installazione, eseguita anch’essa mediante NetBeans, deve essere selezionato come metodo di installazione *Nokia terminal connected via PC Suite*. Così facendo l’IDE utilizza il protocollo della suite per l’invio dei files JAR e JAD al dispositivo, nel quale verranno poi ultimate le operazioni di installazione.

```java
package example.hello;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;


public class HelloWorldMIDlet
    extends MIDlet
{
    // Costruttore della classe HelloWordMIDlet, vuoto.
    public HelloWorldMIDlet()
    {
    }


    // Metodo invocato quando l'applicazione viene istanziata.
    // Dopo l'invocazione l'AMS riserva le risorse necessarie alla sua
    // esecuzione, passando l'applicazione allo stato ACTIVE.
    // Il metodo verifica se esiste un display corrente creando un textScreen
    // se la verifica fosse negativa e settandola come display corrente.
    public void startApp()
    {
        Displayable current = Display.getDisplay(this).getCurrent();
        if(current == null)
        {
            HelloScreen helloScreen = new HelloScreen(this, "Hello World.");
            Display.getDisplay(this).setCurrent(helloScreen);
        }
    }


    // Metodo vuoto, ma che è necessario implementare in una MIDlet
    // La sua eventuale abilitazione va impostata nel file JAD.
    public void pauseApp()
    {
    }

    // Il metodo termina l'applicazione portandola allo stato DESTROYED.
    // L'argomento è un booleano dal quale dipende la modalità di
    // chiusura: false permette all'applicazione di rifiutare la terminazione
    // lanciando un'eccezione, mentre true impone all'AMS di chiuderla in
    // qualunque stato essa dovesse trovarsi.
    public void destroyApp(boolean b)
    {
    }

    // Tale metodo invoca il precedente, permettendo la chiusura
    // dell'applicazione con l'argomento false. Il secondo metodo è utile
    // all'AMS per passare l'applicazione allo stato DESTROYED e per liberare
    // le risorse occupate.
    void exitRequested()
    {
        destroyApp(false);
        notifyDestroyed();
    }
}
```

```
package example.hello;

import javax.microedition.lcdui.*;


class HelloScreen
    extends TextBox
    implements CommandListener
{
    private final HelloWorldMIDlet midlet;
    private final Command exitCommand;

    // Costruttore della classe. Una textBox generalmente ha tre aree:
    // un titolo, un area contenente stringhe, e un'area per i comandi.
    // E' implementata anche la classe CommandListener per permettere
    // l'uso del comando exit da parte dell'utente.
    HelloScreen(HelloWorldMIDlet midlet, String string)
    {
        super("HelloWorldMIDlet", string, 256, 0);
        this.midlet = midlet;
        exitCommand = new Command("Exit", Command.EXIT, 1);
        addCommand(exitCommand);
        setCommandListener(this);
    }


    // Metodo utilizzato per terminare l'applicazione una volta
    // selezionato il comando exit, invocando il comando exitRequest
    // implementato nel precedente file.
    public void commandAction(Command c, Displayable d)
    {
        if (c == exitCommand)
        {
            midlet.exitRequested();
        }
    }
}
```

BIBLIOGRAFIA

Nokia Forum (Internet, consultato il 10-02-10)

Disponibile all'indirizzo: http://www.forum.nokia.com/

Symbian Foundation Community (Internet, consultato il 10-02-10)

Disponibile all'indirizzo: http://www.symbian.org/

Wikipedia (Internet, consultato il 10-02-10)

Disponibile all'indirizzo: http://www.wikipedia.org/

Netbeans IDE (Internet, consultato il 10-02-10)

Disponibile all'indirizzo: http://www.netbeans.org/

Sun Italia (Internet, consultato il 10-02-10)

Disponibile all'indirizzo: http://it.sun.com/