A Giacomo…

Introduzione

Questa tesi di laurea è stata sviluppata presso la ditta Quix s.r.l. di Soliera. Si tratta della realizzazione di un'infrastruttura per il riconoscimento tramite un lettore di codice a barre di ogni unità presente in un magazzino, finalizzata alla gestione automatizzata e in tempo reale della posizione di ogni singola parte. L'applicativo che è stato realizzato, fa parte di un progetto più ampio, che prevede la stampa automatizzata di Documenti di Trasporto, la gestione degli ordini, il monitoraggio in tempo reale della posizione della merce e la gestione logistica del magazzino.

La trattazione proseguirà nel seguente modo: dopo una breve introduzione sul progetto e sulle specifiche imposte dal cliente, seguirà una panoramica sull'ambiente di sviluppo che è stato scelto per la realizzazione del progetto (Capitolo 1).
Dal secondo capitolo in poi, inizia la descrizione dell'applicativo che è stato realizzato in maniera dettagliata, a partire dall'interfaccia grafica per l'utente.
Il terzo capitolo tratta la parte centrale del progetto, ovvero la sincronizzazione tra il palmare e il server; in esso sono anche incluse le descrizioni dei due database (quello sul server e quello sul palmare) e alcune considerazioni per quanto riguarda i tempi di sincronizzazione.
In ordine di importanza, l'argomento successivo alla sincronizzazione col server è la gestione degli errori più comuni, trattata nel quarto capitolo; come sarà specificato più avanti, è fondamentale che l'applicativo preveda e prevenga il maggior numero di errori causati dall'utente, e che non chieda a quest'ultimo di risolvere eventuali errori di sistema.
Infine, il quinto capitolo descrive i test che sono stati effettuati, sia in azienda (in fase di sviluppo) che presso il cliente, e le scelte che questi hanno imposto a fronte dei problemi riscontrati.
Per facilitare la comprensione di termini specifici e poco utilizzati nel linguaggio comune è stato realizzato un breve glossario.

I. Inquadramento del progetto

Il progetto in questione consiste nello sviluppo di un applicativo in grado di riconoscere una parte stoccata in magazzino attraverso la lettura del suo codice a barre, visualizzare le informazioni inerenti la parte stessa, registrare

l'operazione svolta dall'utente (carico, scarico o nulla) e, infine, la sincronizzazione col server. Quest'ultima operazione è la parte centrale del progetto; inizia quando l'operatore inserisce il palmare nella sua base e prevede l'aggiornamento di un database remoto e, successivamente, la copia di questo database sul palmare.

L'applicativo che è stato realizzato lavorerà su diversi palmari, utilizzati dai magazzinieri; i palmari in questione sono degli Intermec™ 700 Color™ (in figura) su cui è installato il sistema operativo Microsoft Pocket PC 2002.

Figura 1: Il Pocket PC Intermec 700 Color

Tutti i dati inerenti le parti stoccate in magazzino sono archiviati in un database residente in un server SQL interno alla ditta. Ogni volta che l'operatore sincronizza il palmare con tale server, alcune tabelle di questo database vengono copiate in un database temporaneo su ogni palmare, in modo che l'operatore possa lavorare con questi dati senza essere costantemente collegato al server. Queste macchine sarebbero, infatti, in grado di sostenere una connessione wireless, ma ciò sarebbe stato piuttosto difficoltoso da gestire da un punto di vista delle norme di sicurezza. Infatti, l'infrastruttura wireless presente all'interno della ditta committente, richiede un'autenticazione attraverso un codice che cambia ogni minuto; questo crea un grave impedimento per l'utente, il quale si troverebbe ad inserire il codice per ogni parte movimentata, senza contare il fatto che l'applicativo deve ridurre al minimo le operazioni svolte dall'utente (si presuppone che l'utenza media abbia una bassissima cultura informatica).

I palmari in questione sono dotati di un dispaly touch-screen e un tastierino numerico con alcuni pulsanti base. Un altro accessorio di cui sono dotate queste macchine è uno scanner laser per la lettura dei codici a barre, elemento fondamentale di questo progetto.

Il modello del palmare su cui è stato realizzato l'applicativo è stato scelto dal cliente, in quanto è stato studiato per applicazioni industriali.

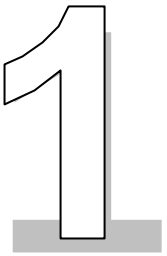
II. Specifiche

Chi ha commissionato il lavoro, ha precisato alcune specifiche di progetto, alcune generate dalle infrastrutture già presenti, altre da necessità quali i limiti degli effettivi utilizzatori del sistema.

Il committente ha chiesto che:

- l'applicativo sia in grado di riconoscere una parte stoccata in magazzino effettuando la scansione del suo codice a barre
- aggiorni lo stato della parte selezionata a seconda dell'operazione scelta dall'utente (carico o scarico)
- l'interfaccia grafica sia essenziale ed estremamente semplice
- non vengano mostrati errori dell'applicazione all'utente, anche se dovuti ad un cattivo utilizzo dell'applicativo, in quanto egli non sarebbe in grado di risolverli
- I dati relativi ad ogni parte del magazzino siano aggiornati in tempi brevi in modo da ottenere un monitoraggio pressoché in tempo reale
- La sincronizzazione col server possa essere effettuata in qualunque momento

Le specifiche sopra indicate non sono particolareggiate, ma indicano soltanto le linee guida che sono state seguite durante la realizzazione del progetto.



Panoramica sull'ambiente di sviluppo

Questo capitolo si propone di fornire qualche informazione di base sull'ambiente di sviluppo che è stato utilizzato per portare a termine la realizzazione di questo progetto. Le informazioni fornite di seguito sono puramente descrittive; per nozioni più approfondite si rimanda ai rispettivi manuali.

Essendo il cliente, per il quale è stato realizzato il software, un partner Microsoft, l'ambiente di sviluppo deve essere adeguato alle sue esigenze e potenzialità; questo significa che la scelta dell'ambiente di sviluppo è condizionata sia da richieste specifiche del committente (ad esempio il sistema operativo sui palmari), sia da potenzialità che questo possiede (essendo egli partner Microsoft non ha nessun tipo di problemi per quanto riguarda le licenze e il reperimento di software a pagamento). L'ambiente di sviluppo che è stato scelto è *Visual Studio .NET 2003*, il linguaggio di programmazione con cui è stato sviluppato il software è *Visual C# .NET 2003*. Per l'accesso e la gestione dei dati, l'applicativo utilizza *Microsoft SQL Server 2000 Windows® CE Edition* (*SQL Server CE*), collegandosi ad un server database implementato con *Microsoft SQL Server*

2000. entrambe queste piattaforme sono ottimamente integrate in *Visual Studio .NET*.
Visual Studio .NET è lo strumento Microsoft di seconda generazione per la creazione e la distribuzione di software *Microsoft .NET*.
Visual Studio .NET 2003 include una versione avanzata di *Windows .NET Framework*, basato sulle precedenti versioni; esso è dotato di nuove funzioni e include miglioramenti sia del software che della documentazione. Grazie al supporto integrato per *.NET Compact Framework*, *Visual Studio .NET 2003* inserisce nell'ambiente *.NET* i dispositivi portatili e incorporati come *Pocket PC*, così come altri dispositivi dotati del sistema operativo *Microsoft Windows CE .NET*.
Visual Studio .NET può essere utilizzato per:

- Creazione di applicazioni per Windows.
- Creazione di applicazioni Pocket PC.
- Creazione di applicazioni Web.
- Creazione di applicazioni Web portatili in grado di riconoscere il dispositivo di destinazione.
- Utilizzo dei Web service in tutti i contesti applicativi sopra indicati.
- Eliminazione dei conflitti noti come "inferno delle DLL".
- Eliminazione dei costosi problemi di distribuzione e di manutenzione delle applicazioni.

Visual Studio .NET è l'unico ambiente di sviluppo realizzato ex novo per consentire l'integrazione con i *Web service*. Attraverso la condivisione dei dati via Internet tra le applicazioni, i *Web service* consentono agli sviluppatori di comporre le applicazioni con codice nuovo ed esistente, indipendentemente dalla piattaforma, dal linguaggio di programmazione o dal modello di oggetti.

1.1 Windows .NET Framework

Windows .NET Framework è il componente di *Microsoft Windows*® per la creazione e l'esecuzione di applicazioni software e *Web service*, componenti che facilitano l'integrazione condividendo i dati e le funzionalità in rete mediante protocolli standard indipendenti dalla piattaforma, quali *XML*, *SOAP* e *HTTP*.
Windows .NET Framework fornisce un ambiente basato su standard per l'integrazione degli investimenti esistenti con applicazioni e servizi della prossima generazione e la flessibilità necessaria per far fronte ai problemi di distribuzione e utilizzo delle applicazioni aziendali.

Windows .NET Framework è costituito da due parti principali: *Common Language Runtime* e una serie unificata di librerie di classi, tra cui *ASP.NET* per le applicazioni *Web* e i *Web service*, *Windows Forms* per applicazioni per client intelligenti e *ADO.NET* per l'accesso a dati a regime di controllo libero.

1.1.1 Il .NET Compact Framework

Il *.NET Compact Framework* è studiato per i dispositivi compatti (pocket PC, Smart phone ecc); rappresenta un elemento fondamentale per lo sviluppo di applicazioni mobili, in quanto mette a disposizione funzionalità per dispositivi intelligenti come il modello di programmazione unificato con *.NET Framework* su desktop e su server, il supporto integrale per gli *XML Web service*, l'accesso ai dati mediante *ADO.NET* e *XML*, e librerie di classi che permettono di realizzare applicazioni sofisticate in tempi ridotti.
Questa piattaforma è supportata da *Microsoft SQL Server 2000 Windows CE Edition version 2.0*. Quest'ultimo è il database relazionale che permette di sviluppare rapidamente applicazioni che estendono la gestione e l'analisi dei dati aziendali anche ai nuovi dispositivi intelligenti. SQL Server CE 2.0 è l'unico database mobile che si integra con .NET Compact Framework permettendo di immagazzinare ed estrarre in locale i dati. Microsoft Visual Studio .NET fornisce un insieme omogeneo di strumenti e interfacce per creare applicazioni attraverso le tecnologie mobili di Microsoft.
Purtroppo, essendo questa piattaforma destinata a dispositivi portatili (quindi con poca memoria), Microsoft ha dovuto ridurre notevolmente le funzionalità del *.NET Framework* per arrivare a questo prodotto (che occupa poco meno di 2 MB di memoria su disco), di conseguenza lo sviluppatore ha a sua disposizione un numero decisamente ridotto di metodi e proprietà per ogni classe.

1.2 Il linguaggio C#

C# (C Sharp) è un linguaggio orientato agli oggetti che consente ai programmatori di creare rapidamente una vasta gamma di applicazioni per la nuova piattaforma *Microsoft .NET*.
Rappresenta un'ottima scelta per la creazione di architetture per una vasta gamma di componenti, dagli oggetti business di alto livello ad applicazioni a livello di sistema. Utilizzando i semplici costrutti di questo linguaggio, è possibile convertire questi componenti in *Web service*, che possono essere richiamati su Internet da qualunque linguaggio in esecuzione su qualsiasi sistema operativo.

Questo linguaggio è, inoltre, progettato per offrire una modalità di sviluppo rapido, senza sacrificare la potenza e il controllo che hanno rappresentato le caratteristiche distintive di *C* e *C++*. Proprio grazie a questa eredità, *C#* presenta un livello di somiglianza molto elevato con *C* e *C++*. Incorpora il supporto per trasformare ogni componente in un *Web service* che possa essere richiamato su Internet da qualsiasi applicazione in esecuzione in qualsiasi piattaforma. Inoltre, l'infrastruttura dei *Web service* può rendere i *Web service* esistenti simili ad oggetti *C#* nativi.
Per migliorare le prestazioni, consente di associare direttamente i dati *XML* ad un tipo di dati strutturati invece che ad una classe. Si tratta di un modo più efficiente per gestire piccole quantità di dati.
Il design di questo linguaggio elimina la maggior parte degli errori di programmazione più comuni in *C++*. Ad esempio:

- Le attività di *garbage collection* sollevano il programmatore dal compito di dover gestire la memoria manualmente.
- In *C#* le variabili vengono inizializzate automaticamente dall'ambiente.
- Le variabili sono indipendenti dai tipi.

Inoltre, integra direttamente nel linguaggio il supporto per la gestione delle versioni. Ad esempio, l'*override* dei metodi deve essere esplicito e non può avvenire inavvertitamente come in *C++* o in *Java*. In questo modo si prevengono gli errori di codifica e si conserva la flessibilità nella gestione delle versioni. Una funzionalità correlata è il supporto nativo per le interfacce e l'ereditarietà delle interfacce. Queste funzionalità consentono di sviluppare e migliorare nel tempo strutture complesse.
Infine, *C#* include il supporto nativo per le *API COM* (*Component Object Model*) e basate su *Windows*® e consente un utilizzo limitato dei puntatori nativi.

1.3 SQL Server e SQL Server CE

Microsoft SQL Server 2000 è la soluzione database e di analisi dei dati che permette di avere buone prestazioni e la scalabilità e l'affidabilità richieste dagli ambienti *Web* e *line-of-business* più esigenti a livello *enterprise*. Il supporto per *XML* e *HTTP* semplifica l'accesso e l'interscambio dei dati, mentre le capacità di analisi danno valore alle informazioni archiviate. Le funzionalità di gestione automatizzano le operazioni di routine e gli strumenti e i servizi di programmazione rendono più rapido il lavoro di sviluppo.
SQL Server 2000 permette di semplificare l'integrazione dei sistemi di *back-end*

e il trasferimento dei dati via *firewall* grazie all'utilizzo delle funzionalità *XML* e al supporto per altri standard Internet come *XPath*, *XSL* e *XSLT*. Gli sviluppatori Web possono accedere ai dati tramite *XML* senza programmare database relazionali, mentre gli amministratori di database possono manipolare i dati nel formato *XML* utilizzando *Transact-SQL* (*T-SQL*) e le procedure archiviate. È anche possibile connettersi via Web a database SQL Server senza programmazione aggiuntiva impiegando una connettività database sicura via *HTTP*. I dati possano essere interrogati, anche da sviluppatori principianti, attraverso un *URL* e interfacce utente intuitive.

Microsoft SQL Server 2000 Windows® CE Edition (*SQL Server CE*) versione 2.0, è un sistema compatto di gestione di database che consente di sviluppare rapidamente applicazioni capaci di estendere ai dispositivi mobili le funzionalità di gestione dei dati aziendali. Si tratta di uno strumento che semplifica lo sviluppo di applicazioni mobili grazie al supporto della sintassi SQL e, inoltre, di un modello di sviluppo e API coerenti con SQL Server.

SQL Server CE rende disponibile un insieme di funzionalità di database relazionale, tra cui un ottimizzatore per le query (*Query Optimizer*) e il supporto per transazioni e tipi di dati diversi, pur mantenendo la compattezza necessaria per preservare risorse di sistema. Grazie a *Remote Data Access* e alla replica di tipo *merge*, i dati disponibili in database di SQL Server vengono recapitati in maniera affidabile e possono essere manipolati offline, per essere sincronizzati con il server in un secondo momento.

SQL Server CE 2.0 è progettato per integrarsi con *Microsoft .NET Compact Framework*, semplificando in tal modo lo sviluppo di applicazioni di database per dispositivi intelligenti. Utilizzando il nuovo provider di dati di SQL Server CE per gestire il codice tramite *Common Language Runtime*, gli sviluppatori di applicazioni mobili possono, infatti, realizzare applicazioni dotate di funzionalità di gestione dati non in linea, particolarmente utili in ambienti non connessi.

SQL Server CE mette a disposizione una serie di potenti funzionalità per la gestione dei dati progettate per i dispositivi mobili. Grazie ad un modello operativo e di programmazione coerente con il resto dei prodotti SQL Server, SQL Server CE può essere facilmente integrato con i sistemi esistenti.

Essendo le risorse di sistema (ad esempio la memoria disponibile) spesso carenti, è fondamentale che un database relazionale sia il più compatto possibile, pur disponendo di tutte le funzionalità essenziali. SQL Server CE rende disponibili queste funzionalità in circa 1 MB di memoria. Ottimizzato anche nelle prestazioni grazie al *Query Processor*, supporta un'ampia gamma di tipi di dati che assicurano una buona flessibilità.

SQL Server CE consente l'accesso ai dati sia tramite dispositivi connessi in

modo permanente che con dispositivi connessi solo saltuariamente al sistema SQL Server. La funzionalità Remote Data Access consente agli utenti remoti di accedere ai dati conservati in database SQL Server 6.5, SQL Server 7.0 e SQL Server 2000 tramite l'esecuzione in remoto di istruzioni Transact-SQL. Utilizzato con SQL Server 2000, SQL Server CE offre inoltre funzionalità per la sincronizzazione tramite replica di tipo merge.

Nonostante queste due piattaforme si integrino piuttosto bene, SQL Server CE non supporta tutti i tipi di dato di SQL Server. Ad esempio, le colonne di testo di tipo non-unicode non sono supportate (*varchar*, *char* e *text*). Inoltre, mentre SQL Server permette una lunghezza di più di 8000 byte per i campi *nvarchar* e *varbinary*, SQL Server CE supporta una dimensione massima di 510 byte. Inoltre, la versione CE non include le funzionalità Analysis Service e Data Transformation Services (DTS).

2

L’interfaccia grafica

L’interfaccia grafica richiesta dal cliente deve essere semplice da utilizzare e d’immediata comprensione, per venire incontro alla bassa cultura informatica degli utenti dell’applicativo, al fine di rendere più efficiente il loro lavoro.
Inoltre è necessario che l’utente non possa in alcun modo accedere ad altro se non all’applicativo in questione; questo si traduce disabilitando controlli quali la chiusura della finestra principale e il menu avvio. Per garantire maggiormente quest’aspetto il programma viene lanciato all’avvio del palmare. Il risultato finale è che, una volta lanciato l’applicativo, non si può utilizzare altro.
Il programma presenta soltanto due form (ognuno con diverse casistiche): un form principale, presente per la maggior parte del tempo d’utilizzo dell’applicativo, e un form per la visualizzazione delle informazioni della parte identificata dal codice a barre letto.
Un aspetto su cui è stata posta particolare attenzione è la posizione del cursore durante l’utilizzo dell’applicazione (*focus*). È indispensabile che il software gestisca autonomamente l’impostazione del *focus* all’interno del form, in quanto l’utente deve poter eseguire il minor numero di operazioni possibile e, durante

l'utilizzo dell'applicativo, si presuppone che egli si curi soltanto della scansione del codice a barre e della scelta se movimentare o meno la parte in questione (qualora essa debba essere movimentata), senza preoccuparsi che il programma sia in condizione di accettare una lettura, ovvero che il *focus* sia impostato sulla casella di testo adibita a questo scopo. Per questi motivi, occorre prevedere tutti i casi dai quali può scaturire uno spostamento del *focus* all'interno del form, quali una scansione andata a buon fine, la pressione accidentale di un qualsiasi punto diverso dalla casella di testo sullo schermo (essendo esso touch-screen) o la pressione di alcuni tasti (o combinazioni di essi) del tastierino del palmare.

2.1 Il Form principale

Il form principale è costituito da una casella di testo compilabile, altre due (non compilabili) per la visualizzazione del numero di parti caricate e scaricate, contrassegnate dalle rispettive etichette, e una label che indica lo stato dell'unità. È inoltre presente un pulsante per forzare la sincronizzazione, che viene reso visibile soltanto quando il palmare è inserito nella base.

Come già specificato in precedenza, sono stati disabilitati tutti i controlli: non sono presenti né il menu avvio né il tasto di chiusura del form. Considerando le forti limitazioni del *.NET Compact Framework* (dovute a problemi di memoria, come meglio specificato nel paragrafo 1.1.1), l'unico modo per ottenere l'eliminazione del menu avvio è impostare il form in modalità *full-screen*, utilizzando metodi contenuti in *aygshell.dll* e *coredll.dll*; l'eliminazione del tasto di chiusura del form è invece possibile anche dal designer di *Visual Studio .NET*.

In figura 2, è mostrata la schermata iniziale. Questa schermata è visibile soltanto quando l'applicazione viene lanciata; essa impone all'utente di sincronizzare il palmare, in quanto ogni volta che viene lanciato il programma, viene cancellato il database temporaneo eventualmente presente sul palmare e ne viene creato uno nuovo (senza alcun dato).

Figura 2: Schermata iniziale

Per poter impostare il *focus*, è necessario abilitare il form, quindi (in teoria) l'utente potrebbe utilizzare il palmare. Di fatto

questa schermata appare soltanto a chi installa l'applicativo o lo lancia per la prima volta. Nel caso il palmare sia appena stato configurato, si presuppone che chi installa l'applicativo, prima di cederlo all'utente finale, esegua una sincronizzazione preliminare. Nel caso, invece, che un utente riuscisse a spegnere e riaccendere il palmare (premendo per un intervallo di tempo superiore a 10 secondi il tasto di accensione/spegnimento), si vedrebbe comparire questa schermata e, se il palmare non viene sincronizzato, ad ogni scansione la risposta dell'applicativo sarebbe "parte non riconosciuta", in quanto il database temporaneo del palmare, in quel momento, è vuoto. In questo modo si evitano errori di utilizzo, ma si ha una perdita di dati elaborati fino a quel momento e non ancora sincronizzati. Questa implementazione è però stata ben accettata dal cliente, in quanto, tutto sommato, la possibilità che si verifichi un evento del genere è piuttosto remota, dal momento che anche togliendo la batteria (grazie ad una batteria di backup) il palmare rimane praticamente acceso, ovvero quando la batteria viene inserita, lo stato della macchina è lo stesso in cui era al momento dell'estrazione.

Figura 3: Sincronizzazione

Quando l'operatore ha finito il suo lavoro, ripone il palmare nella base e inizia la sincronizzazione. Come mostrato in figura 3, il form viene completamente disabilitato e lo sfondo diventa rosso. Questa particolarità del colore di sfondo è stata richiesta espressamente dal cliente, al fine di favorire la comprensione dello stato del palmare da parte degli utenti. Inoltre viene visualizzato il tasto *refresh* (disabilitato durante la sincronizzazione), il quale servirà per forzare una successiva sincronizzazione.

Durante l'operazione di sincronizzazione (che dura circa due minuti e trenta secondi), è indispensabile che l'utente non estragga il palmare dalla sua base; oltre al colore dello sfondo, quindi, l'utente ha come ulteriore segnale un messaggio che gli indica che l'unità non è operativa, e che non deve utilizzare il palmare. Nel caso l'utente non faccia attenzione a questi accorgimenti ed estragga il palmare, l'applicativo entra in un caso di errore e non cede il controllo del palmare fintanto che non viene effettuata una nuova sincronizzazione.

Una volta terminato il processo di sincronizzazione, l'aspetto del form cambia e diventa quello mostrato in figura 4. Il tasto *refresh* viene abilitato e lo sfondo

Figura 4: Sincronizzazione avvenuta

diventa giallo, per indicare all'utente che la sincronizzazione è avvenuta e da questo momento può utilizzare il palmare; tuttavia, fintanto che il form resta giallo, la sincronizzazione può avvenire soltanto premendo il tasto *refresh* e non semplicemente inserendo la macchina nella base. Qualora l'operatore scelga di effettuare una nuova sincronizzazione (premendo *refresh*), il form torna rosso come in figura 3. Come nel caso precedente, la label in basso indica lo stato del palmare, per sottolineare le operazioni che può effettuare l'utente.

Quando il palmare viene estratto dalla sua base, l'applicativo rimane in questo stato per un intervallo di tempo variabile da pochi secondi a circa un minuto, dopodiché l'unità diventa completamente operativa, e passa nello stato di funzionamento normale.

In figura 5 è mostrata la schermata di funzionamento normale, ovvero quando l'unità è completamente operativa. Il tasto *refresh* viene nascosto e lo sfondo diventa bianco, per indicare all'utente che può utilizzare la macchina liberamente. Da questo momento l'operatore può effettuare scansioni di codici a barre (e di conseguenza decidere se movimentare o meno una determinata parte) o sincronizzare il suo palmare col server, semplicemente inserendolo nella base.

Figura 5: Unità operativa

Nel caso un utente legga un codice non inserito nel database o con una codifica errata, viene visualizzato un breve messaggio di errore del tipo "*parte sconosciuta*" o "*codice non previsto*".

L'operatore può inserire un codice da far riconoscere all'applicativo sia premendo il tasto di scansione laser, presente sul lato destro del palmare, sia inserendolo a mano col tastierino numerico e poi premendo il tasto *Action* o *Enter* (sulla tastiera). In entrambi i casi, se la lettura è andata a buon fine, l'applicativo visualizza un nuovo form (paragrafo successivo) contenente le informazioni inerenti alla parte selezionata.

2.2 Il Form delle informazioni

Figura 6: Informazioni parte

Ogni volta che viene riconosciuta dall'applicativo una parte stoccata in magazzino, viene visualizzato un nuovo form (figura 6), per visualizzare le informazioni della parte e permettere all'operatore di scegliere se movimentarla o meno. La scelta può essere effettuata sia premendo sui pulsanti del form, che sui tasti della tastiera corrispondenti (esempio *Esc* o *Action*). Qualora la parte non sia da movimentare, vengono visualizzate soltanto le informazioni, ed un unico tasto per chiudere il form.

Come per il form principale è stato disabilitato il menu avvio, ma, in questo form, il tasto di chiusura è stato lasciato attivo; la chiusura tramite questo tasto equivale a premere il tasto *no*.

Premendo un tasto qualsiasi di questo form, l'applicativo chiude il form, dopodiché, nel caso l'operatore abbia scelto di movimentare la parte, aggiorna il database e, infine, cede il controllo al form principale, ritornando nella schermata di funzionamento normale.

3

La sincronizzazione tra il palmare e il server

La sincronizzazione tra il palmare ed il server è la parte cruciale di questo progetto. È necessario che essa avvenga in modo sicuro e senza errori, cercando di venire incontro alle esigenze del cliente.
Va inoltre premesso che l'applicativo sarà utilizzato in due sedi differenti: nella prima opereranno cinque o sei utenti, mentre nella seconda sarà presente un solo operatore. Prima di iniziare la progettazione e lo sviluppo di questa parte, è stato necessario effettuare qualche verifica sull'infrastruttura di rete presente all'interno della ditta committente in entrambe le sedi.
L'infrastruttura di rete presente nella prima area di utilizzo è piuttosto vasta; è dotata di un server DHCP, un server DNS e WINS e diversi server SQL, dove risiede anche il database per questo applicativo. Dal momento che il server database risiede nella stessa rete di utilizzo dei palmari, non sussistono problemi di autenticazione.
Nell'altra area, invece, l'infrastruttura di rete è decisamente meno estesa e, soprattutto, il palmare è costretto ad accedere ad un server che non si trova nella stessa rete, pertanto necessita di autenticazione; per motivi di sicurezza e

di praticità, il cliente vuole evitare che l'operatore debba autenticarsi ad ogni sincronizzazione. Occorre quindi che il palmare si connetta ad un server locale il quale, a sua volta, si connetterà autenticandosi al server database.

Da quanto appena detto si deduce facilmente che il tipo di connessione e la procedura di sincronizzazione per le due aree sarà differente sotto alcuni aspetti. In entrambi i casi, comunque, nonostante le limitazioni imposte dalla struttura di rete, è importante fare in modo che l'utilizzo dell'applicativo sia il medesimo.

3.1 La connessione tra il server e il palmare

Per far fronte alle esigenze di ogni area, sono state analizzate (e poi implementate) due diverse procedure di sincronizzazione. L'utilizzo dell'applicativo da parte dell'utente è rimasto il medesimo, ovvero per effettuare la sincronizzazione egli deve soltanto riporre il palmare nella base (o premere il tasto *refresh* qualora la macchina sia già inserita nel suo supporto). L'unica differenza sta nel tempo complessivo di sincronizzazione, il quale verrà analizzato nel dettaglio nel paragrafo 3.4.
Per entrambe le procedure, le operazioni de eseguire sono le medesime, cambia soltanto la loro implementazione. Ogni volta che si richiede una sincronizzazione, l'applicativo:

- Aggiorna il database remoto
- Elimina il database locale
- Lo ricrea (vuoto)
- Lo popola secondo i nuovi dati del database remoto

La prima procedura è quella più sicura ed efficace, ovvero il software effettua la sincronizzazione attraverso una transazione SQL. Questo sistema prevede la modifica dei dati riga per riga, la quale, però, avviene effettivamente soltanto al momento della conferma, cioè quando viene invocato il comando *commit*.
La seconda implementazione, invece, è basata sullo scambio di file XML. Quando ha inizio la procedura di sincronizzazione, il software genera un file XML dal database locale e lo invia al server; a questo punto copia un altro file XML, appositamente generato dal server, sulla memoria del palmare, dal quale estrae i dati aggiornati e popola il suo database.

Il primo metodo è indubbiamente il migliore, in quanto, oltre a garantire la

sicurezza del trasferimento dei dati, permette al palmare di essere autonomo, ovvero l'unica condizione per cui possa sincronizzarsi è che il server al quale tenta di connettersi sia acceso. La seconda procedura, invece, richiede che sul server sia presente una routine che controlli la presenza di modifiche sul file XML e generi un altro file XML con i dati aggiornati periodicamente (quest'aspetto non è stato realizzato in quanto esula da questa parte di progetto). Infine, come già accennato poco sopra, tra le due implementazioni c'è una notevole differenza di tempo: nel primo caso la durata complessiva dell'operazione è di circa due minuti e trenta secondi, mentre nel secondo può arrivare anche a più di cinque minuti. Questi tempi sono comunque stati accettati dal cliente, in quanto nella seconda area di utilizzo (dove si usa l'implementazione con XML) l'operatore è soltanto uno e la mole di lavoro è decisamente inferiore da permettere tempi di attesa più lunghi.

3.2 Il database temporaneo sul palmare

L'utilizzo delle piattaforme e dell'ambiente di sviluppo scelto permettono di creare un vero e proprio database SQL sul palmare. Ovviamente, sempre per motivi di spazio, non è completo quanto un database normale (ad esempio non ci sono viste, ruoli ecc), ma tutte le funzioni di base sono implementate. Per realizzare l'applicativo è stato creato un database di prova (in quanto il database definitivo è direttamente dipendente da quello remoto che, al momento, non è ancora stato realizzato); esso è costituito da due tabelle, una contenente tutti i dati delle parti che devono essere movimentate (*movimentazione*) e una in cui sono archiviati tutti i dati di tutte le parti stoccate in magazzino (*anagrafica*). La prima tabella cambia frequentemente, ed è costituita (previsione sul funzionamento dell'applicativo a regime) da circa 150 record, mentre per la seconda si prevedono circa 2000 righe. La chiave primaria per ogni tabella è un numero riprodotto dal codice a barre applicato su ogni singola parte; inoltre, ogni record prevede campi che descrivono il responsabile del reparto a cui appartiene la parte, il giorno di registrazione sul database, una breve descrizione, il luogo di partenza e quello di arrivo. Nella tabella *movimentazione*, è previsto un campo (di default impostato a 0) nel quale possa essere inserito un numero identificativo del palmare che ha effettuato l'operazione di movimentazione della parte corrispondente; infine, sempre in questa tabella, vi è un campo che indica lo stato di ogni singola parte (questo campo può assumere cinque valori differenti: da caricare, da scaricare, caricato, scaricato, da non movimentare).

Il linguaggio C# permette di creare ed utilizzare con estrema semplicità un piccolo database locale. Di seguito è descritta una piccola procedura di esempio per creare e popolare un database come quello di prova che è stato realizzato. L'attenzione di questa trattazione non è focalizzata tanto sul codice che implementa la procedura, quanto sull'algoritmo e le scelte progettuali che sono state svolte.
C# utilizza il concetto di *namespace* che corrisponde (a grandi linee) al più noto concetto di *package* per Java. Le namespaces dove risiedono le classi per l'utilizzo di databases sono *SqlClient* e *SqlServerCe*.
Per creare un database locale occorre istanziare un oggetto della classe *SqlCeEngine* e poi invocare il metodo *CreateDatabase()*:

```
SqlCeEngine engine = new SqlCeEngine("Data Source = LocalDataBase.sdf");
engine.CreateDatabase();
```

A questo punto il database è stato creato; per poter lavorare su di esso è necessario creare, ed aprire, una connessione verso tale database e, a partire dalla connessione (oggetto della classe *SqlCeConnection*), creare un'istanza di un comando SQL con il quale effettuare le operazioni di creazione delle tabelle e popolazione del database (oggetto della classe *SqlCeCommand*).

```
m_conn = new SqlCeConnection("Data Source = LocalDataBase.sdf");
m_conn.Open();

m_cmd = m_conn.CreateCommand();
```

Il testo del comando appena istanziato, rappresenta il comando SQL vero e proprio che si vuole eseguire. La creazione di una nuova tabella risulterà quindi essere:

```
m_cmd.CommandText = "CREATE TABLE Movimentazione"
                  + "(Codice_Parte int,"
                  + "Responsabile ntext,"
                  + "Descrizione ntext,"
                  + "Registrato ntext,"
                  + "Partenza ntext,"
                  + "Arrivo ntext,"
                  + "ID_Partenza int,"
                  + "ID_Arrivo int,"
                  + "Movimentato int)";

m_cmd.ExecuteNonQuery();
```

Si noti che soltanto l'invocazione del metodo *ExecuteNonQuery()* crea la tabella sul database, mentre la prima parte di questo frammento di codice imposta

semplicemente una proprietà del comando per la creazione di una tabella.

3.2.1 L'aggiornamento del database temporaneo

Il database temporaneo sul palmare viene aggiornato (o più in generale modificato) per due diversi motivi: la movimentazione di una parte o la sincronizzazione col database remoto.
Per quanto riguarda il primo caso, ogni volta che l'operatore scansiona (o inserisce a mano) un codice a barre, l'applicativo lo ricerca nel suo database e, se lo trova, visualizza le informazioni inerenti. A questo punto, se la parte è da movimentare e l'operatore decide di farlo, il software modifica il campo della tabella movimentazione in cui è indicato lo stato dell'oggetto e aggiorna anche il dataset su cui sono memorizzati i dati (quest'ultimo passaggio serve per la sincronizzazione attraverso file XML).

```
m_ds.Tables[0].Rows[i][9] = (int)state;

m_cmd.CommandText = "UPDATE Movimentazione SET Movimentato = "
      + (int)state + " WHERE Codice_Parte = " + readedBarcode;

m_cmd.ExecuteNonQuery();
```

Questa procedura è la stessa sia per la sincronizzazione tramite transazione SQL che per l'implementazione con XML.
Per quanto riguarda, invece, il secondo caso, ovvero l'aggiornamento del database temporaneo causato dalla sincronizzazione col server, la trattazione si divide in due parti, una per ogni tipo di implementazione della procedura di sincronizzazione.

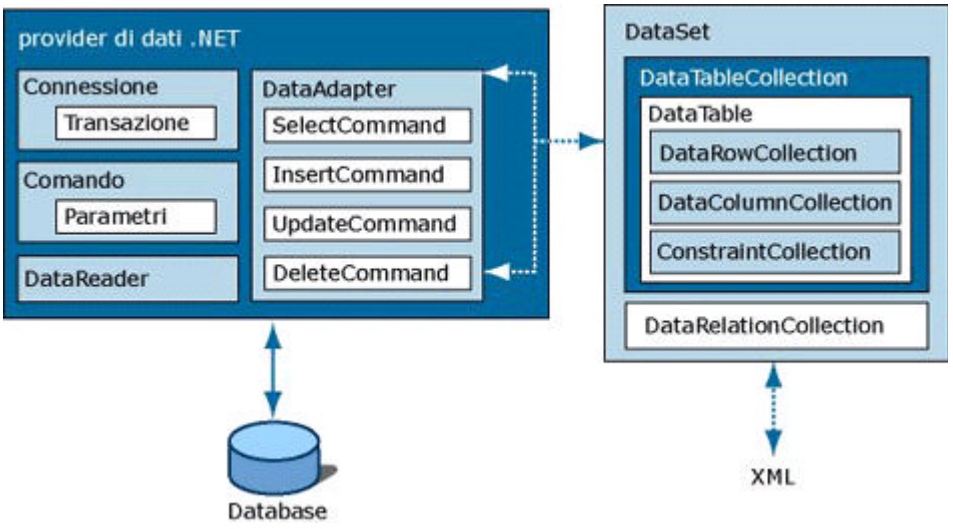


Figura 7: Utilizzo di DataAdapter e DataSet

Nel caso la sincronizzazione avvenga tramite transazione SQL, dopo che è stato aggiornato il database remoto, il software inizia ad aggiornare quello locale. Il linguaggio C# fornisce una classe (la classe *SqlDataAdapter*) per interfacciarsi con un database SQL; l'utilizzo di questa classe e della classe *DataSet* è schematizzato in figura 7; attraverso questa classe è possibile importare facilmente e velocemente dati in un *DataSet*. La classe *DataSet* permette di lavorare con una serie di dati in modo strutturato; di fatto, è come lavorare con un database, senza, però, memorizzazione persistente. Una volta che i dati sono archiviati all'interno del DataSet, è possibile accedervi specificando il nome (o l'indice) della tabella a cui ci si riferisce e selezionando un determinato campo (o record) considerando la tabella come se fosse una matrice.
La schematizzazione di figura 7 è valida anche se il database in questione è quello sul palmare, ovvero un database SQL CE; l'implementazione con DataAdapter non è stata utilizzata per la realizzazione di questa parte dell'applicativo, in quanto si è ritenuto sufficiente il supporto fornito dalle classi *SqlCeConnection* e *SqlCeCommand*. L'utilizzo del DataAdapter diventa molto utile in casi come questo, dove la connessione con il database remoto deve essere il più breve possibile.
La classe SqlDataAdapter permette di accedere ad un database in modo semplice ed affidabile; è possibile specificare la connessione da utilizzare, quattro tipi di comando (in modo che ogni volta che si richiede l'esecuzione di una serie di istruzioni, si possa impostare come si comporta il DataAdapter. Come mostrato in figura, è possibile popolare un DataSet a partire dai dati contenuti in un database interfacciandosi con un oggetto della classe SqlDataAdapter. Questa operazione avviene in modo veloce e sicuro semplicemente invocando il metodo *Fill(Dataset,NomeTabella)*. Il frammento di codice qui riportato mostra come popolare due tabelle di un DataSet utilizzando questo metodo.

```
m_extConn.Open();

m_da = new SqlDataAdapter();

m_da.SelectCommand = new SqlCommand();
m_da.SelectCommand.Connection = m_extConn;
m_da.SelectCommand.CommandText = "SELECT Codice_Parte,"
      + "Responsabile,Descrizione,Registrato,"
      + "Partenza,Arrivo,ID_Partenza,ID_Arrivo,Movimentato FROM
Movimentazione";
m_da.Fill(m_ds,"Movimentazione");

m_da.SelectCommand.Connection = m_extConn;
m_da.SelectCommand.CommandText = "SELECT * FROM Anagrafica";
m_da.Fill(m_ds,"Anagrafica");
```

Per quanto riguarda la procedura di sincronizzazione attraverso file XML, popolare il DataSet risulta essere ancora più semplice. La struttura della classe DataSet è, infatti, basata su XML, di conseguenza esiste già implementato il metodo *ReadXml(NomeFile)*, come mostrato dal seguente frammento di codice.

```
File.Copy(m_sharedFolder + "database.xml","database.xml",true);
m_ds.ReadXml("database.xml");
```

Una volta popolato il DataSet, con l'una o l'altra procedura, occorre importare i dati nel database locale. La soluzione più semplice è quella di eseguire una serie di INSERT per ogni riga di ogni tabella del DataSet. Questa soluzione non è forse la più ortodossa, ma è certamente la più sicura.

```
for(int i=0;i < m_ds.Tables[1].Rows.Count;i++)
{
      m_cmd.CommandText = "INSERT INTO Anagrafica" +
      " (Codice_Parte,Responsabile,Descrizione," +
      "Registrato,Partenza,Arrivo) VALUES ("
            + m_ds.Tables[1].Rows[i][0] + ", '"
            + m_ds.Tables[1].Rows[i][1] + "', '"
            + m_ds.Tables[1].Rows[i][2] + "', '"
            + m_ds.Tables[1].Rows[i][3] + "', '"
            + m_ds.Tables[1].Rows[i][4] + "', '"
            + m_ds.Tables[1].Rows[i][5] + "')";
      m_cmd.ExecuteNonQuery();
}
```

Il popolamento del database potrebbe anche essere fatto direttamente dal database remoto, ma questa implementazione è stata scartata quasi subito in quanto la scrittura in memoria richiede parecchio tempo e per riportare una tabella da circa 2000 record verrebbero impiegati circa due minuti, tempo nel quale si potrebbe incorrere in un errore grave qualora l'operatore estragga il palmare dalla base; con l'implementazione presentata, invece, il tempo di sincronizzazione è circa il medesimo (tempo totale di sincronizzazione, vedi paragrafo 3.4 per un'analisi più dettagliata), ma l'intervallo di tempo "a rischio" scende a circa 15 secondi.

3.3 L'aggiornamento del database remoto

L'aggiornamento del database remoto avviene in due modi diversi, uno per ogni procedura implementata. Di seguito vengono descritti entrambi i metodi nel dettaglio.

Per quanto riguarda la prima implementazione, la parte di aggiornamento del database remoto è forse la più importante e delicata. Al momento della sincronizzazione il palmare inizia la *transaction*, ovvero comincia ad aggiornare tutti i record del database remoto che corrispondono a quelli che sono stati modificati sul database locale (cioè quelli delle parti movimentate), dopodiché esegue il *commit*. Nel caso qualcuno interrompa la connessione durante il trasferimento dei dati (ossia prima che l'applicazione esegua il *commit*), ad esempio sollevando il palmare dalla sua base, nessun record del database sul server viene modificato, in quanto l'applicativo lancia una eccezione ed esegue un *rollback*, cioè ripristina i dati sul database col valore che avevano prima dell'inizio della transazione.

```
m_extConn.Open();

m_extCmd = new SqlCommand();
m_extCmd.Connection = m_extConn;
m_trans = m_extConn.BeginTransaction("MyTransaction");
m_extCmd.Transaction = m_trans;

try
{
      ...

      m_trans.Commit();
}
catch(Exception ex)
{
      m_trans.Rollback("MyTransaction");
}
```

Soltanto nel caso in cui la transazione vada a buon fine, il software elimina il database sul palmare, lo ricrea (vuoto) e poi lo popola estraendo i nuovi dati da quello remoto. Se, invece, la sincronizzazione non va a buon fine, l'applicativo non cede il controllo all'operatore (fatta eccezione del tasto *refresh*) e richiede una nuova sincronizzazione.

L'altra procedura è sicuramente più semplice da implementare, ma molto più onerosa sul lato server, in quanto è necessario un altro applicativo per il monitoraggio del file XML. Generare un file XML da un database si è dimostrato piuttosto semplice. Dopo che l'oggetto DataSet è stato istanziato e sono stati salvati dei dati su di esso, è possibile generare un file XML contenente tutti questi dati (e mantenendo la stessa struttura) invocando il metodo *WriteXml(NomeFile)*. A questo punto il palmare può copiare il file così generato sul server, il quale si occuperà della gestione dei dati e del vero e proprio aggiornamento del database remoto.

```
m_ds.WriteXml("database.xml");
File.Copy("database.xml", m_sharedFolder + "database.xml",true);
File.Delete("database.xml");
```

Dopo che il file appena generato è stato copiato nella directory condivisa sul server, viene cancellato. Eseguita quest'operazione l'applicativo copierà il nuovo file XML dal server dal quale popolerà il database.

3.4 Analisi dei tempi di sincronizzazione

Dal momento che esistono due diverse implementazioni per la sincronizzazione tra il palmare ed il server, è necessario effettuare una breve analisi sui tempi impiegati dall'applicativo per eseguire le operazioni. Purtroppo, viste le forti limitazioni presenti sul palmare (in questo caso a livello di piattaforma), non è possibile effettuare misurazioni con precisione migliore di un secondo.
Il test che è stato effettuato prevede cinque sincronizzazioni consecutive; sono stati misurati i tempi effettivi di sincronizzazione e i tempi di abilitazione, ovvero il tempo impiegato dall'applicativo a cedere il controllo completo all'utente.

Prima di eseguire l'analisi vera e propria, occorre specificare un particolare riguardo alla procedura di sincronizzazione.
Come già accennato e come mostrato dalla figura 8, il palmare effettua l'effettivo trasferimento dei dati del database soltanto nella prima parte della procedura di sincronizzazione; il tempo rimanente (in figura visualizzabile dove non c'è attività di rete) è impiegato per popolare il database temporaneo.
La fine del processo di sincronizzazione è evidenziata in figura dal riprendere dell'attività di rete; questo scambio di dati è dovuto al metodo adottato dal palmare per la verifica della presenza di una connessione di rete. È importante che i dati vengano scambiati col server nel minor tempo possibile per evitare errori di aggiornamento. Se, infatti, l'operatore decidesse di estrarre il palmare dalla base prima che gli venga indicato di farlo, potrebbe interrompere la connessione creata col server; se la procedura fosse stata implementata senza l'utilizzo del DataAdapter, la connessione col server avrebbe dovuto durare per quasi tutto il tempo di sincronizzazione totale, aumentando notevolmente i rischi di errore. La figura 8 si riferisce alla procedura di sincronizzazione effettuata con l'implementazione tramite transazione SQL; il grafico per l'altro tipo di sincronizzazione è molto simile, con l'unica differenza che il trasferimento dei dati dura qualche secondo di meno.

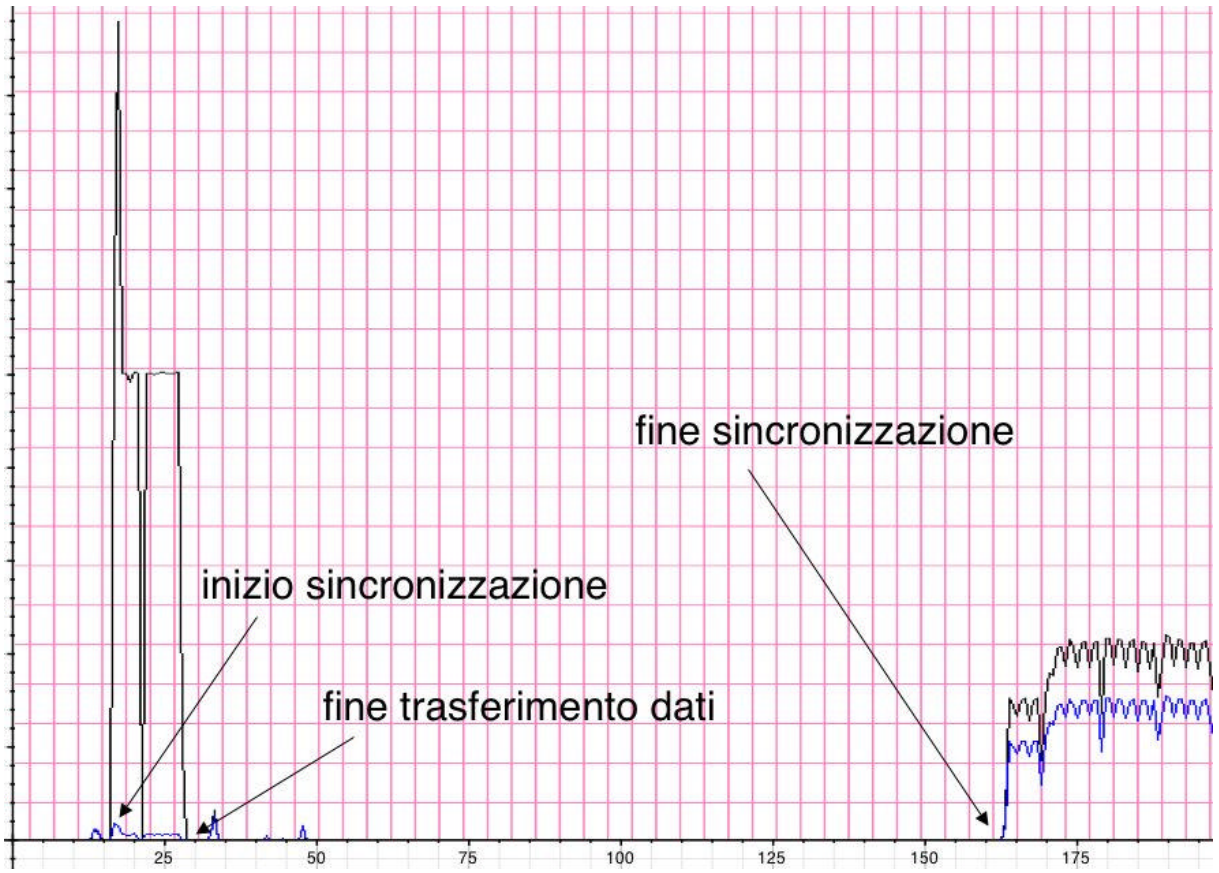


Figura 8: Sincronizzazione - Traffico sulla rete

Di seguito saranno messi a confronto i tempi di sincronizzazione effettiva e di abilitazione del palmare. Per ogni tipologia di sincronizzazione è stata calcolata anche una media per avere un valore indicativo da presentare al committente.

La tabella riportata di seguito fa riferimento al tempo di effettiva sincronizzazione, ovvero da quando l'operatore appoggia il palmare sulla base a quando il form diventa giallo (vedi capitolo 2). Il valore percentuale dell'ultima colonna esprime il tempo guadagnato effettuando la sincronizzazione tramite transazione SQL piuttosto che con un file XML.

	tempo di sincronizzazione		tempo guadagnato (percentuale)
	XML	SQL	
prima volta	05:11,0	01:43,0	66,88%
primo refresh	04:23,0	01:51,0	57,79%
secondo refresh	05:14,0	01:37,0	69,11%
terzo refresh	04:44,0	01:55,0	59,51%
quarto refresh	04:40,0	01:43,0	63,21%
Media	04:50,4	01:45,8	63,57%

Tabella 1: Tempi di sincronizzazione effettivi

Come evidenziato anche dal grafico, sotto questo aspetto, l'implementazione tramite transazione SQL permette di ottenere delle prestazioni nettamente più convenienti.

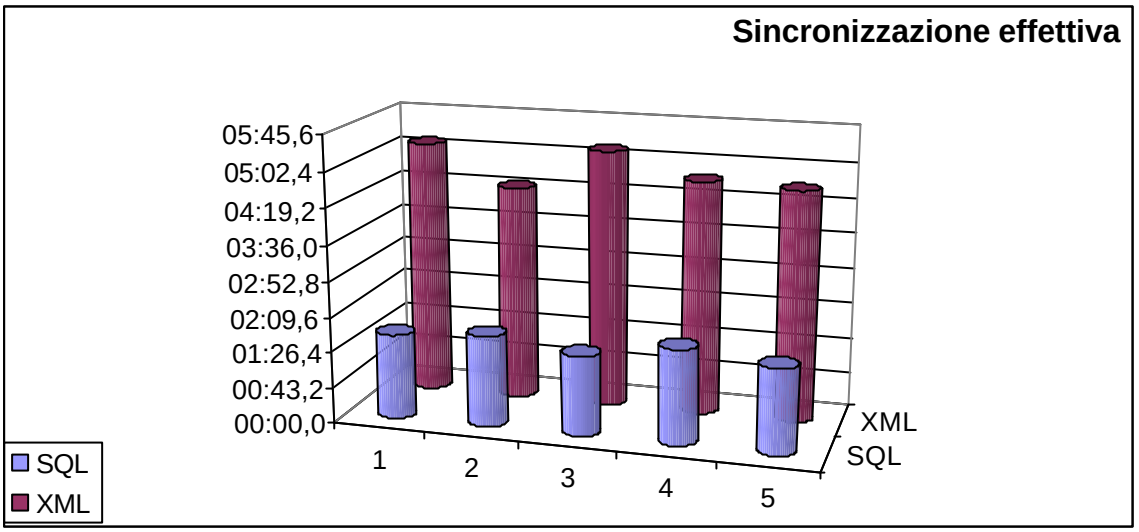


Figura 9: Tempi di sincronizzazione effettiva

L'entità elevata del tempo di sincronizzazione nell'implementazione con file XML è dovuta all'operazione di lettura dall'XML e popolazione del DataSet, operazione che occupa il palmare per circa tre minuti. Anche il tempo necessario a popolare il database non è trascurabile (più o meno un minuto), ma è comune ad entrambe le implementazioni, pertanto non rilevante ai fini del confronto.
L'altra misura che è stata effettuata è la valutazione del tempo di abilitazione dopo una sincronizzazione, ossia del tempo che occorre al palmare prima di verificare l'assenza del segnale di rete e cedere il controllo completo all'operatore. Nell'ultima colonna è indicata la percentuale di tempo che viene perso utilizzando l'implementazione con transazione SQL piuttosto che quella con XML. La grossa differenza di tempo è probabilmente dovuta al metodo di riconoscimento della presenza del segnale di rete; esso è basato sulla verifica dell'esistenza di un file contenuto nella stessa directory del file XML, quando il software non trova il file, significa che la connessione di rete non è più attiva, pertanto l'applicativo cede il controllo all'operatore.

	tempo di abilitazione		tempo perso (percentuale)
	XML	SQL	
prima volta	00:02,0	00:58,0	96,55%
primo refresh	00:19,0	00:58,0	67,24%
secondo refresh	00:02,0	00:58,0	96,55%
terzo refresh	00:02,0	00:58,0	96,55%
quarto refresh	00:02,0	00:58,0	96,55%
Media	00:05,4	00:58,0	90,69%

Tabella 2: Tempi di abilitazione

Il grafico seguente mostra i risultati di tabella 2.

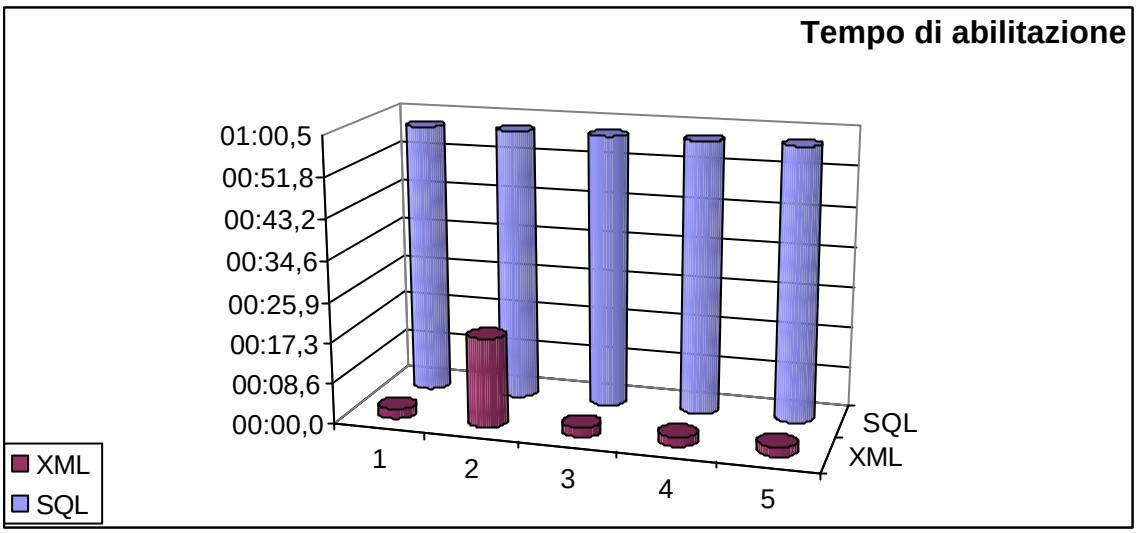


Figura 10: Tempi di abilitazione

Nella tabella e nel grafico successivi sono riassunti i risultati ottenuti nelle due misure.

	totale sincronizzazione		tempo guadagnato (percentuale)
	XML	SQL	
prima volta	05:13,0	02:41,0	48,56%
primo refresh	04:42,0	02:49,0	40,07%
secondo refresh	05:16,0	02:35,0	50,95%
terzo refresh	04:46,0	02:53,0	39,51%
quarto refresh	04:42,0	02:41,0	42,91%
Media	04:55,8	02:43,8	44,62%

Tabella 3: Tempi totali di sincronizzazione

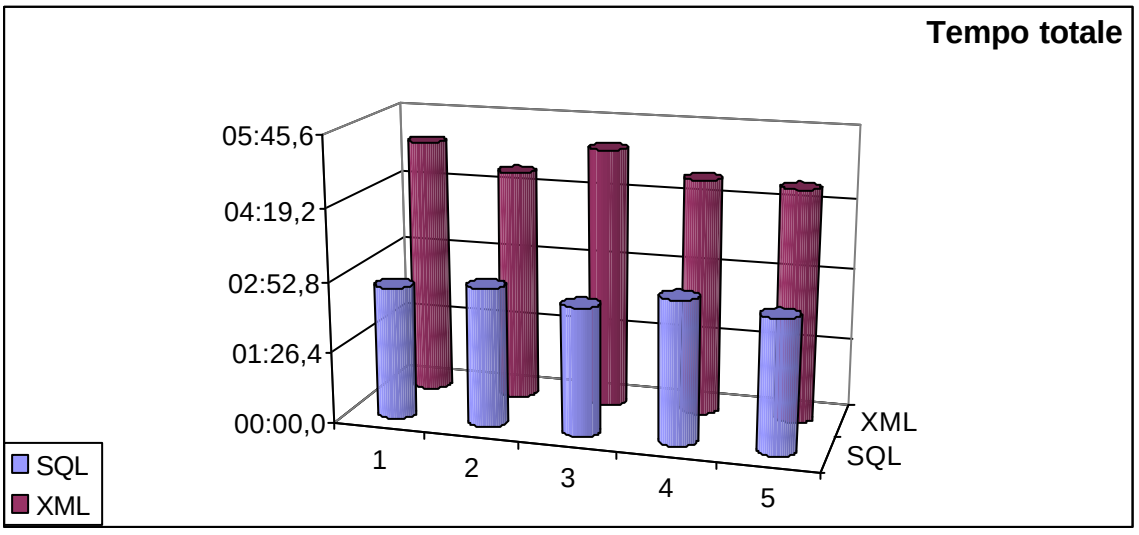


Figura 11: Tempi totali di sincronizzazione

Complessivamente si può dire che l'implementazione che utilizza la transazione SQL è comunque migliore, in termini di tempo. Infatti, nonostante sia più lenta del 90% circa rispetto all'altra nell'abilitazione del palmare, è comunque più veloce nella sincronizzazione effettiva, pertanto il tempo totale da quando l'operatore inserisce il palmare nella base a quando è completamente operativo dopo che lo ha estratto è migliore del 45% circa.

La variabilità dei dati dipende soprattutto dalla stabilità della larghezza di banda ottenibile dalla rete a cui si connettono i palmari e non tanto da quante parti essi abbiano movimentato (cioè dal numero di righe da aggiornare sul database remoto).

In conclusione si può affermare che per quanto possibile si tenderà ad utilizzare la transazione SQL, sia per ragioni d'affidabilità, che per ragioni di tempo. Purtroppo, come già spiegato, questo non è sempre possibile, si sta pertanto cercando di trovare una soluzione alternativa per l'area di utilizzo dove è necessaria l'autenticazione, al fine di garantire una sincronizzazione veloce e sicura come quella utilizzata dall'altra area.

La gestione degli errori più comuni

È stato espressamente richiesto dal cliente che gli errori siano gestiti tutti autonomamente senza mai richiedere l'intervento dell'utente per risolverli.
Questo capitolo vuole illustrare tutti i casi di errore nei quali si può incorrere durante l'utilizzo dell'applicativo. Generalmente, in caso di errore, l'applicazione non fa nulla, al massimo si limita ad inibire il controllo e richiedere una sincronizzazione.
Gli errori in cui si può incorrere sono di due tipi: errori generati dall'utente ed errori di sistema. I primi sono quasi sempre da attribuirsi ad un cattivo utilizzo dell'applicativo, mentre i secondi possono essere generati in diversi modi (ad esempio il server SQL spento…).

4.1 Casi di errore sul palmare

In questo paragrafo saranno descritti nel dettaglio i casi d'errore generati sul palmare. Nonostante si sia cercato di prevedere tutti gli errori possibili, potrebbe

essere che, compiendo determinate azioni oppure al verificarsi di determinate circostanze, venga generato un errore che non era stato contemplato né in fase di progettazione, né in fase di sviluppo. Per evitare *crash* dell'applicazione, ogni metodo è stato incluso in un blocco *try-catch* che cattura qualsiasi eccezione; con questo tipo d'implementazione, però, non è immediatamente identificabile di quale tipo di errore si tratti, pertanto è stato necessario includere le istruzioni che potrebbero generare errori prevedibili in blocchi *try-catch* specifici. Ogni volta che un'eccezione viene catturata, l'applicativo genera un messaggio di errore (soltanto se necessario) e lancia l'eccezione catturata al chiamante (qualora esista).

4.1.1 Errori generati dall'utente

Tutti gli errori generati dall'utente sono causati dall'estrazione precoce del palmare dalla base, fatta eccezione dello spegnimento della macchina (che causa la chiusura forzata dell'applicativo).
Se l'operatore solleva il palmare dalla base mentre sta effettuando la sincronizzazione, può generare due tipi di errore: durante l'aggiornamento o durante la lettura dei dati dal database remoto. Ogni volta che si incorre in uno di questi due errori, il software mostra all'utente un messaggio di errore standard ("Errore! Sincronizzazione non riuscita:") a cui aggiunge il messaggio specifico, a seconda dell'errore avvenuto.

Errore durante l'aggiornamento del database remoto: accade quando l'operatore solleva il palmare mentre il software sta aggiornando il database remoto.
Se si sta lavorando con l'implementazione che usa la transazione SQL, i problemi sono minimi, in quanto viene eseguito il *rollback* e nessuna riga del database remoto viene modificata; a questo punto il palmare interrompe la procedura di sincronizzazione, visualizza il messaggio di errore e richiede all'operatore una nuova sincronizzazione, inibendo tutti i controlli (fatta eccezione del tasto *refresh*). Il messaggio di errore specifico è: "impossibile aggiornare il DB remoto".
Se, invece, si sta utilizzando l'implementazione con file XML, il palmare termina la procedura e avvisa l'utente dell'errore, imponendo, come nel caso precedente, la re-sincronizzazione. La differenza rispetto a prima risiede sul server: infatti, nella directory condivisa, sarà presente un file, rappresentante il database del palmare, incompleto; se la routine in esecuzione sul server tentasse di leggerlo per poter aggiornare il database remoto, incorrerebbe sicuramente in un errore. È quindi necessario che l'operatore che utilizza

l'applicativo in questa modalità (che è uno soltanto) sia adeguatamente istruito e provveda ad una nuova sincronizzazione quanto prima. Il messaggio visualizzato sarà: "impossibile scrivere il file xml".

Errore durante la lettura del database remoto: questo errore si verifica quando il palmare viene estratto dalla base durante la lettura dal database remoto.
Nel caso la lettura avvenga tramite DataAdapter, l'errore che si verifica è piuttosto grave; il database locale risulta incompleto e non è possibile nessuna operazione su di esso. Il software impedisce all'utente di eseguire qualsiasi operazione se non una nuova sincronizzazione e visualizza il messaggio di errore: "impossibile leggere il DB remoto".
Se al momento dell'estrazione l'applicativo stava copiando il file XML, invece, l'errore è meno grave, in quanto il database non è incompleto, ma vuoto. Anche in questo caso viene richiesta una nuova sincronizzazione e visualizzato il messaggio di errore: "impossibile leggere il file xml".

Gli errori generati dall'utente sono un numero decisamente ridotto. Questo è reso possibile da una serie di controlli che vengono effettuati in molte parti del programma. Ad esempio, la lettura del codice a barre tramite lo scanner laser elimina tutto l'eventuale contenuto della casella di testo inserito accidentalmente. Un altro controllo che viene effettuato sull'input è che tutti i caratteri inseriti siano numeri: nel caso siano presenti caratteri, il software provvede ad eliminarli e a considerare soltanto la parte rimanente della stringa.
È necessario anche aggiungere che la probabilità che si verifichi questo tipo di errori è sufficientemente bassa, in quanto l'operazione di aggiornamento e lettura del database remoto, per quanto riguarda l'implementazione con transazione SQL e DataAdapter, dura circa 15 secondi, mentre per la procedura con XML, lo scambio dei file è ancora più veloce (poco meno di 10 secondi).

4.1.2 Errori di sistema

Gli errori di sistema sono meno prevedibili di quelli generati dall'utente e, soprattutto, è più difficile evitarli effettuando dei controlli simili a quelli descritti nel paragrafo precedente. Essi possono dipendere da errori del sistema operativo (*crash*), da errori di configurazione (ad esempio il file di configurazione setup.ini è stato impostato in maniera errata) o da errori di rete (server SQL spento o guasti nella rete). Anche in questo caso il palmare visualizza un breve messaggio di errore e spesso richiede una nuova sincronizzazione.

Errore durante la lettura del file di configurazione: al fine di permettere una certa flessibilità di utilizzo dell'applicativo, è stato creato un file di configurazione (qui a fianco ne è mostrato un esempio) dal quale il software può caricare alcuni parametri, quali il nome del database, il nome del server, la directory condivisa ecc. quando il programma viene lanciato, cerca immediatamente di leggere questo file al fine di impostare i parametri in esso contenuti. Qualora il file di configurazione sia assente o non leggibile, si incorre in un errore che non permette all'applicazione di proseguire (non essendo impostato nessun nome del server, non è possibile effettuare alcuna sincronizzazione, pertanto, nonostante l'applicazione non venga chiusa, non è possibile effettuare alcuna operazione). Se si verifica questo errore, viene visualizzato il messaggio "can't read from setup.ini file". L'unico modo per risolvere questo tipo di errore è verificare che il file in questione esista o che sia compilato correttamente.

```
[setup]
confirmRequested=yes
transaction=yes

[server]
name=SERVER
sharedFolder=share

[database]
name=TestDataBase
table0=Movimentazione
table1=Anagrafica
username=TestDataBaseUser
password=TestDataBaseUser

[localMachine]
ID=1234567890
```

Errore durante l'apertura della connessione verso il database remoto: gli errori di connessione non sono generati soltanto dall'utente, ma possono essere causati anche da errori di configurazione o di rete. Infatti, sia quando l'applicativo deve aggiornare il database remoto che quando deve leggere i suoi dati per copiarli su quello locale, cerca di aprire una connessione verso tale database. Questa operazione potrebbe fallire, ovvero si potrebbe incorrere in un errore, per diversi motivi:

- La stringa di connessione non è corretta, ovvero il file di configurazione setup.ini è stato compilato in maniera errata.
- Il server SQL è spento o, comunque, non attivo.
- Il palmare non riesce a raggiungere il server SQL a causa di un guasto sulla rete.

In ogni caso il palmare visualizza il messaggio di errore in cui afferma l'impossibilità a leggere o scrivere sul database remoto e chiede una nuova sincronizzazione. Nel secondo o terzo caso (server spento o guasto), una nuova sincronizzazione potrebbe risolvere il problema, in quanto la rete o il server potrebbero essere stati ripristinati, mentre nel primo una nuova sincronizzazione genera sicuramente un errore identico e, pertanto, è necessario modificare il file di configurazione. Questo tipo di errore, può in-

correre soltanto nel caso si utilizzi l'implementazione con transazione SQL.

Errore durante l'aggiornamento del database remoto: la possibilità che si verifichi questo errore per cause non dipendenti dall'operatore, è alquanto remota; infatti, esso incorre se, mentre l'applicativo sta aggiornando il database remoto, o sta copiando nella directory condivisa il file XML, il server si spegne oppure si guasta la rete proprio in quel momento. Anche in questo caso il palmare visualizza il messaggio di errore e impone una nuova sincronizzazione.

Errore durante l'aggiornamento del database locale: questo errore avviene per le stesse cause del precedente, con l'unica differenza che incorre quando l'applicativo sta leggendo i dati dal database remoto o sta copiando in locale il file XML dalla directory condivisa. Il comportamento in questo caso è identico a quello dell'errore precedente.

Errore durante il popolamento del database locale: se i dati caricati nel DataSet non sono validi (ad esempio manca un campo in un record), l'applicativo, al momento del popolamento del database locale, incorre in un errore e interrompe la procedura, visualizzando il solito messaggio di errore ("impossibile popolare il DB locale") e richiedendo nuovamente la sincronizzazione. Questo tipo di errore incorre principalmente quando si utilizza l'implementazione con XML, in quanto è più probabile che il server generi un XML errato dal suo database piuttosto che si abbia un errore nel riempimento del DataSet utilizzando il metodo *Fill*.

Errore nel riconoscimento della presenza di una connessione di rete: quello descritto di seguito non è un vero e proprio errore, nel senso che l'utente non si accorge dello stesso se non dal fatto che, inserendo il palmare nella base, la sincronizzazione non avviene automaticamente. L'errore è imputabile ad una errata configurazione del file setup.ini. Il metodo utilizzato per il riconoscimento della presenza di una connessione di rete, infatti, è basato sulla verifica dell'esistenza di un file contenuto nella directory condivisa sul server: se il software verifica che la directory esiste, significa che è disponibile una connessione di rete (ovvero che il palmare è inserito nella base), altrimenti il palmare è considerato off-line. A questo punto, se la directory indicata dal file di configurazione è errata, la verifica dell'esistenza di un file in tale cartella dà sempre esito negativo; di conseguenza non viene mai riconosciuto l'evento di inserimento del palmare nella base e la

sincronizzazione non può mai avvenire. Per questo tipo di errore non viene visualizzato nessun messaggio e l'applicativo non fa assolutamente nulla, in quanto non è neanche in grado di accorgersene. L'unico modo per risolvere questo errore è correggere il file di configurazione e lanciare nuovamente il programma; ovviamente, tutti i dati elaborati fino al quel momento andranno persi.

Da quanto appena detto, risulta evidente che chi si occuperà della compilazione del file di configurazione (presumibilmente un incaricato della gestione del sistema all'interno della ditta committente), presti molta cura nello svolgere questo tipo di attività.

4.2 Risoluzione automatica degli errori

La risoluzione automatica degli errori è uno degli argomenti su cui sono focalizzati i prossimi sviluppi del progetto. Attualmente, infatti, non si può parlare di una vera e propria risoluzione automatica degli errori, ma piuttosto di riconoscimento dell'errore. Questo è comunque molto importante, nel senso che, riconoscendo il tipo di errore accaduto, il software può comportarsi in maniera differente e mostrare diversi tipi di messaggio all'utente, in modo che, qualora quest'ultimo non riesca a risolverlo, il responsabile tecnico della ditta committente possa individuare agilmente l'origine del problema.

5

Test dell'applicazione

In questo capitolo saranno descritti alcuni test che sono stati effettuati e i problemi che sono stati riscontrati. I test sono stati effettuati sia in fase di sviluppo che presso la ditta committente.
La descrizione dell'applicativo effettuata fino a questo punto, si riferisce alla versione definitiva, completa di tutte le modifiche apportate alla luce dei test effettuati e dei problemi riscontrati. In questa sezione sarà possibile vedere com'è stata modificata l'implementazione e le scelte effettuate a fronte dei problemi riscontrati.

5.1 Problemi tecnici dovuti ai limiti della piattaforma

Come già detto più volte, la piattaforma utilizzata ha dei limiti dovuti alle dimensioni della memoria presente sul palmare. Per questo motivo non è possibile sfruttare tutte le potenzialità del linguaggio di programmazione o è necessario fare uso di espedienti per risolvere alcune tipologie di errori. Di

seguito saranno illustrati i principali problemi riscontrati durante lo sviluppo dell'applicativo e la fase di testing svolta in azienda.

Connessione tra desktop e palmare: il primo passo per lo sviluppo dell'applicazione è stato permettere al palmare di comunicare con un PC per poter importare i file sorgente da eseguire. L'unico modo indicato dal produttore per comunicare tra palmare e un PC è utilizzando il software MS ActiveSync, già installato sul palmare in ROM. Occorre quindi installare il medesimo software sul desktop. ActiveSync è scaricabile dal sito MSDN. Il problema si è verificato in questo momento: la versione fornita per il download dal sito è la 3.5; nei dettagli del download, Microsoft afferma che questa versione va bene per tutti gli utenti Windows 98 o successivi. Il collegamento tra palmare e PC è stato effettuato tramite porta USB; il problema si è riscontrato in questa operazione: ActiveSync non riusciva a connettersi con il Pocket PC. Consultando gli articoli tecnici sul sito Microsoft, si scopre che, per gli utenti NT (quindi Windows NT, 2000 e XP), fino alla versione 3.7 il collegamento è effettuabile soltanto tramite porta seriale. Per risolvere questo problema, è stata rilasciata la versione 3.7.1. Una volta aggiornata la versione di ActiveSync, il palmare e il PC possono comunicare tra loro.

Chiudere i thread utilizzando il .NET Compact Framework: il .NET Compact Framework è privo di alcuni metodi e proprietà di alcune classi. Per questo motivo, anche la classe *Thread* risulta molto penalizzata; infatti, non sono supportati metodi quali *Abort*, *Suspend*, *Resume* e parecchi altri. Nasce ora un grave problema: come fermare un thread avviato? Per risolvere questo problema si fa uso di un piccolo espediente. Occorre innanzitutto dichiarare due variabili globali: *numThreads* e *closeRequested*. La variabile *numThreads* viene incrementata di 1 ogni volta che viene lanciato un nuovo thread. Il passo successivo è definire un metodo che venga invocato quando si tenta di chiudere l'applicazione (di seguito un esempio):

```
protected override void OnClosing(CancelEventArgs e)
{
      if(this.numThreads > 0)
            e.Cancel = true;
      else
            e.Cancel = false;

      this.closeRequested = true;
}
```

In questo modo la variabile *closeRequested* viene impostata a *true*.
Il codice presente nel metodo che viene invocato quando si lancia il thread deve essere incapsulato in un blocco *do-while* che abbia come condizione (`!this.closeRequested`); quando la variabile *closeRequested* diventa vera, l'applicazione esce dal *do-while* ed esegue l'istruzione successiva (`this.Invoke(new EventHandler(this.CloseMe));`) che invocherà un gestore degli eventi puntato su un metodo che provvederà alla chiusura del thread (compreso quello principale) e all'aggiornamento della variabile *numThreads* (di seguito un esempio):

```
private void CloseMe(object o, EventArgs e)
{
      this.numThreads--;
      this.Close();
}
```

In questo modo, prima di chiudere l'applicazione, vengono chiusi anche tutti i thread aperti.

Verifica della presenza di una connessione di rete: inizialmente la verifica della presenza di una connessione di rete veniva effettuata tentando di risolvere il nome della macchina su cui risiede il database remoto. Se l'applicativo riesce a risolverlo, significa che è disponibile una connessione di rete, altrimenti va in eccezione e ripete nuovamente il tentativo finché quest'ultimo non va a buon fine. Questa implementazione sembra semplice e funzionale, ma, in fase di testing, si riscontra un problema: il palmare conserva in cache l'associazione hostname-indirizzo, quindi il palmare non ha modo di verificare quando viene disconnesso (ovvero quando l'operatore lo estrae dalla base per utilizzarlo), non concedendo mai il controllo all'utente. Inoltre se viene premuto il testo refresh quando il palmare è scollegato (ma non ancora attivo) inizia la procedura di sincronizzazione, la quale, ovviamente, non andrà mai a buon fine. Procedendo nella fase di testing, si è provato a cambiare il nome fisico della macchina con il suo indirizzo interno o con il suo IP; nel primo caso il problema permane, nel secondo, invece non sempre la risoluzione va a buon fine (ovvero a volte riesce a risolvere l'indirizzo ed altre no).
La soluzione che si è quindi scelto di attuare è quella di verificare l'esistenza di un file contenuto all'interno della directory condivisa sul server: se il file esiste, la connessione di rete è disponibile, altrimenti il palmare è scollegato.
Per quanto riguarda l'analisi dei tempi, sono stati riscontrati dei pro e dei contro: il primo metodo che era stato implementato si accorgeva quasi subito

della presenza della rete (meno di un secondo) e iniziava immediatamente la sincronizzazione; il secondo, invece, impiega poco meno di 20 secondi. Il tempo di sincronizzazione effettivo rimane, ovviamente, invariato. Quando invece l'utente estrae il palmare dalla sua basetta, nel primo caso (se l'associazione non rimane in cache) il tempo che trascorre prima della completa operatività è circa 30 secondi, mentre nel secondo caso è piuttosto variabile, anche se spesso si sono riscontrati tempi migliori (dipende da quanto il palmare rimane nella base dopo la fine della sincronizzazione).

5.2 Test presso il cliente

Oltre ai test in fase di sviluppo, sono stati effettuati alcuni test presso la ditta committente, sia per mostrare al cliente il lavoro svolto fino a quel momento (e quindi verificare se le sue esigenze erano state soddisfatte), sia per verificare il funzionamento dell'applicativo nel luogo in cui verrà effettivamente utilizzato.
Di tutti gli incontri con il cliente (principalmente finalizzati alla raccolta delle specifiche e alla dscussione di modifiche da apportare al software), due di questi sono stati dedicati ad effettuare test.

Durante il primo test, è stato riscontrato un serio problema: difficoltà del palmare ad ottenere il lease da parte del server DHCP. In fase di sviluppo non si ha avuto modo di sperimentare questo aspetto, in quanto l'indirizzo IP del palmare era sempre stato impostato staticamente. Durante tutta la durata del test non vi è stato modo di risolvere questo problema.
In ogni caso, è stata presentata l'interfaccia grafica, alla quale sono state richieste alcune modifiche:

- Sostituire i Pop-Up con delle Form e mostrare comunque le informazioni della parte, anche se deve essere movimentata. Infatti, nella prima versione, l'applicativo mostrava un Pop-Up all'utente al momento del riconoscimento di una determinata parte, chiedendogli se movimentare o meno la parte (senza mostrare informazioni su di essa) oppure visualizzando i dati della parte riconosciuta.
- Mettere i due bottoni del Form delle informazioni (conferma e annulla) in corrispondenza dei tasti ACTION ed ESC del tastierino del palmare, in modo da uguagliarne il funzionamento e permettere un utilizzo più immediato dell'applicativo.
- Nel momento della sincronizzazione con il server, occorre far diventare tutto il form rosso, non solo il semaforo. Nella prima versione, il Form

principale restava sempre bianco, mentre cambiava il colore di un rettangolo posizionato in basso a sinistra, il quale svolgeva la funzione di un piccolo semaforo. Per rendere più immediata la comprensione dell'operazione che sta svolgendo il palmare, si è pensato di colorare tutto il Form.

In questa sede, è stato valutato anche il problema d'infrastruttura di rete presente nell'altra area di utilizzo; si è pensato, quindi, all'implementazione con XML e ad inserire un parametro nel file di configurazione che possa permettere di scegliere tra l'una e l'altra.

Per far fronte al problema riscontrato (mancato ottenimento del lease), è stato creato in azienda un ambiente di test in modo da simulare l'infrastruttura di rete presente presso il cliente. Non trovando alcuna soluzione, si è pensato che il problema fosse causato da un errore del sistema operativo verificatosi in precedenza sull'impostazione dei parametri di rete nelle chiavi di registro, pertanto è stato effettuato un hard-reset del palmare, al fine di ripristinare le condizioni iniziali (infatti l'hard-reset prevede l'eliminazione di tutto ciò che non è stato salvato in ROM, ottenendo una configurazione del palmare come fornito dal produttore). Questa soluzione si è verificata efficace, in quanto il palmare (dopo l'hard-reset) è riuscito ad ottenere il lease dal server DHCP.

Nel secondo incontro dedicato ai test presso il cliente, si è però riscontrato un altro problema: il palmare riusciva sì ad ottenere il lease, ma non comunicava il suo Hostname al server. Il cliente è particolarmente interessato a questo aspetto, in quanto desidera tener traccia di chi ha svolto le operazioni di movimentazione della merce. In questo incontro, è stato inoltre deciso di introdurre una colorazione intermedia, dello sfondo del form principale, tra il rosso (palmare in sincronizzazione da non estrarre dalla base) e il bianco (unità operativa), la cui realizzazione è illustrata nel capitolo 2.

Continuando i test in azienda, si è giunti alla conclusione che il sistema operativo installato sui palmari utilizza un protocollo di rete diverso da quello di una qualsiasi macchina Windows, pertanto esso non è "visibile" in rete, né tantomeno identificabile tramite il suo hostname. Per risolvere l'esigenza di monitoraggio delle operazioni richiesta dal cliente, si è introdotto un codice identificativo del palmare nel file di configurazione, il quale sarà inserito in un'apposita colonna del database remoto, come meglio descritto nel terzo capitolo.

6

Conclusioni

L'applicativo che è stato realizzato per lo sviluppo di questa tesi di laurea, permette agli utenti che lo utilizzeranno (i magazzinieri) di rendere più semplice ed efficiente il loro lavoro. Infatti, essi possono conoscere le informazioni di ogni parte presente in magazzino semplicemente leggendo il codice a barre applicato sull'etichetta, ovvero premendo un solo tasto.

Questo tool permette anche di evitare gli errori di spedizione più comuni, in quanto per ogni parte è specificato se è da consegnare e a chi, in modo che l'utente spedisca soltanto la merce che è stata selezionata dall'ufficio direttivo del magazzino, il quale aggiornerà il database remoto a dovere.

Inoltre permette all'azienda committente (e ai suoi clienti) di avere un monitoraggio costante e quasi in tempo reale della posizione della merce, in modo di conoscere in anticipo eventuali errori ed evitare equivoci con i clienti.

Questo tipo di applicazione è studiato per magazzini di grandi dimensioni, in quanto non sarebbe sensato pensare di istituire un'infrastruttura così onerosa come costi (sia d'installazione che di manutenzione) senza averne l'effettiva necessità.

Personalmente ritengo la realizzazione di questo progetto particolarmente formativa, in quanto mi ha permesso di sviluppare conoscenze con tecnologie di ultima generazione e in fase di grande sviluppo (i palmari), ma ancora poco conosciute nell'ambito della programmazione e dello scambio di dati.
Inoltre ho avuto modo di estendere le mie conoscenze informatiche, apprendendo un uovo linguaggio di programmazione (C#) e l'utilizzo di un ambiente di sviluppo piuttosto diffuso presso le aziende, quale Microsoft Visual Studio .NET; questo è stato possibile anche grazie all'apporto di personale competente, il quale mi ha aiutato a risolvere tutti quei problemi che si riscontrano durante l'approccio con un uovo mondo.
Infine, grazie allo sviluppo di questo progetto, ho potuto effettuare una vera esperienza lavorativa, la quale penso mi permetterà un perfetto inserimento nel mondo del lavoro.

7

Bibliografia

Per la stesura di questa tesi di laurea sono stati utilizzati soprattutto e-books e documentazione on-line, elencati di seguito assieme ai libri consultati.

Libri

[1] D. Beneventano, S. Bergamaschi, M. Vincini: *Progetto di Basi di Dati Relazionali: lezioni ed esercizi*, Pitagora editrice – Bologna

[2] Microsoft Corporation: *Microsoft Visual C# .NET Deluxe Learning Edition-Version 2003*, Microsoft press

e-books

[3] Anders Hejlsberg, Scott Wiltamuth: *C# Language Reference*

[4] Microsoft Corporation: *Introduction to C# Programming for the Microsoft® .NET Platform*

[5] Eric Butow, Tommy Ryan: *C#*, manGraphics® & Hungry Minds™

[6] Fernando G. Guerrero, Carlos Eduardo Rojas: *Microsoft SQL Server 2000 Programming by example*, Que®
[7] Patrick Dalton, *Microsoft SQL Server Black Book*, The Coriolis Group
[8] Thuan L. Thai, Hoang Lam: *.NET Framework Essentials*, O'Reilly

Documentazione on-line

[9] *Microsoft Windows Server System – Microsoft SQL Server*; disponibile su: http://www.microsoft.com/italy/sql/Default.mspx
[10] *Microsoft Windows Server System – Microsoft SQL Server – SQL Server CE*; disponibile su: http://www.microsoft.com/italy/sql/ce/default.asp
[11] *Welcome to the .NET Compact Framework QuickStart Tutorial*; disponibile su: http://samples.gotdotnet.com/quickstart/CompactFramework/
[12] *Microsoft .NET Compact Framework Technical Articles*; disponibile su: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetcomp/html/CompactfxTechArt.asp
[13] *Network Programming in C#*; disponibile su: http://www.c-sharpcorner.com/Network/NetworkProgramPart1RVS.asp
[14] *.NET 247: SHFullscreen*; disponibile su: http://www.dotnet247.com/247reference/msgs/34/172882.aspx
[15] *Developing and Deploying Pocket PC Setup Applications*; disponibile su: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetcomp/html/netcfdeployment.asp
[16] *Remote ActiveSync FAQ*; disponibile su: http://www.cewindows.net/wce/20/ras.htm
[17] *Database programming using ADO .NET with C#*; disponibile su: http://www.c-sharpcorner.com/Database.asp
[18] *Designing a Winform in C# and Linking It to a SQL Server Database*; disponibile su: http://www.codeguru.com/Csharp/Csharp/cs_network/database/article.php/c6017
[19] *Example source to connecting to SQL 2000 from C# .NET Application*; disponibile su: http://dbforums.com/arch/30/2002/12/640331
[20] *Visual C# .it – Corso di ADO .NET*; disponibile su: http://www.visualcsharp.it/corsoc.asp
[21] *SQL Server Forums at SQLTeam.com – SQL Server 2000 XML*; disponibile su: http://www.sqlteam.com/Forums/topic.asp?TOPIC_ID=9336&FORUM_ID=5&CAT_ID=3&To

[22] *Inserting relational data using DataSet and DataAdapter*; disponibile su: http://www.codeproject.com/cs/database/relationaladonet.asp

[23] *Data synchronization: which technology?*; disponibile su: http://www.devx.com/Intel/Article/17264/0/page/2

[24] *.NET 247: SQL Server connect Beat 1* ; disponibile su: http://www.dotnet247.com/247reference/msgs/17/88657.aspx

[25] *Know Dot Net - Compact Framework - Use XML Files to Replace The Registry*; disponibile su: http://www.knowdotnet.com/articles/cfcsetting.html

[26] *Know Dot Net - Why Migrate - .NET Eases The Task of Accessing the Registry*; disponibile su: http://www.knowdotnet.com/articles/registrywrapper.html

[27] *A simple architecture to read arbitrarily formatted flat files into ADO DataSets*; disponibile su: http://www.codeproject.com/cs/database/inifileado.asp

[28] *Creating a P/Invoke Library*; disponibile su: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetcomp/html/PInvokeLib.asp

[29] *Working with registry on Pocket PC. Pocket PC Developer Network*; disponibile su: http://www.pocketpcdn.com/articles/registry.html

8

Glossario

Di seguito sarà fornita una breve definizione per i termini specifici e tecnici utilizzati, in modo di facilitare al lettore la lettura di questo trattato.

ADO.NET: è la nuova tecnologia per l'accesso ai dati introdotta da Microsoft con l'avvento di Visual Studio .NET. Essa è basata completamente sulla tecnologia *XML*, in modo che i dati possano essere facilmente consultati da qualsiasi piattaforma.

Blocco try-catch: si tratta di un costrutto che incapsula un blocco di istruzioni che possono generare errori (*eccezioni*). Con il comando catch le eccezioni lanciate non generano errori di sistema, ma possono essere gestite dallo sviluppatore.

Codice a barre: l'utilizzo del codice a barre si sta diffondendo sempre più nell'ambito della catalogazione e riconoscimento automatico di prodotti; esso è basato sul susseguirsi di una serie di linee verticali: una certa sequenza di queste linee rappresenta un carattere, che può essere codificato da uno *scanner laser*.

DHCP: Dynamic Host Configuartion Protocol; è un protocollo di rete che fornisce un meccanismo per assegnare dinamicamente gli indirizzi IP e i parametri di configurazione ad un host tramite il protocollo TCP/IP.

Display touch-screen: si tratta di un display con il quale è possibile interagire toccando lo schermo e non soltanto da tastiera (o altra periferica di input).

Eccezione: è un oggetto che viene generato da una istruzione quando incorre un errore; un'eccezione contiene tutte le informazioni dell'errore che l'ha generata (spesso visualizzabili tramite un messaggio).

File .ini: questo tipo di file serve per configurare alcuni parametri di un'applicazione. Spesso è utilizzato in alternativa alle chiavi di registro.

Focus: questo termine si intende la posizione del cursore all'interno di un *form*. Impostare il *focus* su una casella di testo significa posizionare il cursore su di essa, in modo che la pressione di un tasto da tastiera corrisponda alla visualizzazione di un carattere nella casella di testo.

Form: è una finestra di dialogo in cui l'utente può modificare alcuni parametri (ad esempio compilare caselle di testo).

Lease: letteralmente significa contratto di affitto, contratto di locazione; esprime un "contratto" tra il server e il client *DHCP*, caratterizzato da una durata variabile, che certifica la validità delle impostazioni di rete assegnate ad un determinato host. Per tutta la durata del *lease*, lo stesso host riceverà le stesse impostazioni di rete.

Namespace: è un insieme di classi, strutture e delegazioni che hanno lo stesso ambito o svolgono funzioni simili.

Palmare: è un computer di dimensioni ridotte (sia fisiche che a livello di memoria e prestazioni) utilizzato per gli scopi più diversi: agenda, organizzatore di appunti, applicazioni industriali (come in questo caso) ecc.

Refresh: letteralmente significa rinnovare; in ambito informatico è spesso utilizzato per indicare l'aggiornamento di una pagina web o, come in questo caso, l'aggiornamento di risorse (quali un database).

Scanner laser: è un dispositivo in grado di leggere un codice a barre. È costituito da un raggio laser che viene spostato ripetutamente da destra a sinistra e viceversa, in modo che possa leggere la sequenza di linee che costituiscono il codice a barre.

Sincronizzazione: nell'ambito di questo progetto esprime il processo che viene avviato dall'utente per aggiornare il database sul palmare e quello remoto sul server, al fine di poter lavorare off-line.

Thread: è un processo che, spesso, lavora in parallelo con l'applicazione principale; in questo progetto è stato utilizzato un thread che verifica costantemente la presenza di una connessione di rete.

Web service: è una applicazione che viene eseguita in remoto; è molto utile per fornire servizi che utilizzano applicazioni senza che l'utente debba installare software sulla propria macchina.

WINS: Windows Internet Naming Service; è un servizio che mappa gli indirizzi IP ai nomi di computer NetBIOS e viceversa. Utilizzando i server WINS le risorse possono essere ricercate in base al nome computer invece che con l'indirizzo IP.

XML: eXtensible Markup Language; è un linguaggio che è stato creato per il trasporto dei dati sulla rete. Su questo linguaggio è stato realizzata la piattaforma ADO.NET per l'accesso ai dati in Visual Studio .NET.

Indice

A conclusione di questo lavoro, desidero ringraziare la prof.ssa Sonia Bergamaschi, che ha dimostrato di essere disponibile in qualunque momento e mi ha offerto l'opportunità di fare questa esperienza, l'ing. Massimiliano Malaguti e tutti i soci e dipendenti di Quix, che mi hanno accompagnato in questi tre mesi di stage e aiutato nella realizzazione d questa tesi, dimostrando una disponibilità fuori dal comune.

Ringrazio i miei genitori Ermanno e Lia, che mi hanno sostenuto economicamente e, soprattutto, con il loro affetto in questi anni di università e non solo; un grazie anche al nonno, il quale mi ha insegnato ad accettare la vita con semplicità, a mia sorella Anna e ai miei fratelli Benedetto e Raffaele, senza dei quali, nonostante tutto, non potrei fare.

Ringrazio anche tutti i miei amici: Marco, Cico e Alle, che al grido di "siamo giovani" ne abbiamo fatte tante, Acci, mio fedele compagno di sventura in questi ultimi giorni di università, Rapa, Sonno, Turri, Ste, Effe, Lucia, la Pallina, la Mala, Vale; gli ingegneri: Gianni, Thomas, Sergio, Macco, Beppe Azzarito, Sola, Balbo, Giorgio, Melz, Lasa, Tromba, Anna, la Gianna, Vladi, Franco, Mattia, Canna, Bagatti, Gio Morrone, Fanga, Giuseppe, Francesco, la Vivi, Gioia, Antonio, Pasquale, Stefano, Beppe, Antonio Luciani e tutti i compagni di corso; quelli della BSI: Albertone (che odia il mio mouse), Annama, Marcella, Liliane e Marlyse, Anna F., Giuliotta, Mazzone, Prencipe, Vitulo, Assana, Sophie, Eri; l'ex 5°F: il Nanno, Nicola, Roby, Mone, Luca, Casy, Gozzano, Manni, Zelloni, Guido, Mazzi, Mari, Olio, lo Gnappo (muorbido), Rossi, Tiziano, e gli altri; e ancora: il Giuss, Poggi e la Barbarina, K e la Milly, Elvin, Valeria, Bosi, Pigghi, Martina, Betto, Rubbio (la famigghia!), Dino, Giulia, Rita, Ivan, Agnese (che se sta leggendo questa tesi sa che mi deve una cena), Paolo, Mimmo, Rosario, Freccia-Minguzzo, Antonino, Cerf, Stefanone, Marcello, Richi Guidetti, Fra, lo Snello, Gridi, Greta, Fabia, Paola, Dani Fila, Lorenz, Jessica, Licia, Carmen, Chiara, Isa, Evelina, Nadia, Elena, la Shoes,

Davide, Gio Bell, Emma, Betta e Fede Teggi, Laura, Aiello, la Gigia, Ines, Biagione, Giovi, Gigetto e la Ciccia, Dido la Sammy e la Michy, Erika, Jack, la Fava, Giangi, Fillo, la Patty, i gemelli Canalini Fabrizio e Federico; Angela, Ettore, Piccy, Serra, Alex e tutti i pizzarini di Pizza in Piazza; Gabbro e l'Anna, il Cec, Piola, Ciù, Ciccio, la Magna e la Cri, Coma, Stanza, Cate, Dani Montanari, la Frency e tutti gli amici che, un po' per la fretta, un po' perché è la prima volta che mi laureo, ho dimenticato di scrivere in questo elenco.

Grazie anche ad Antonio, Susi e Sara, che mi hanno sempre accolto per quello che sono.

Un ringraziamento particolare va a Monica, la quale non si è mai stancata di sopportarmi, dimostrandomi ogni giorno tutto l'affetto di cui è capace spingendomi avanti quando volevo mollare.

Andrea