INDICE *Pag.*

1. Introduzione

Questa tesi è il risultato di un'esperienza di stage presso un'azienda che produce e commercia un software gestionale per piccole e medie imprese. Risulta importante oggi l'inserimento in un ambiente di lavoro organizzato sia a livello personale, sia per l'azienda, che in qualche modo cerca di avere al proprio interno team di persone affiatato e che interagiscano e lavorino in modo efficiente. Nel mio caso si tratta di un esperienza assolutamente nuova, che mi ha permesso di vedere quale fosse il mio grado di adattamento e le problematiche che sorgono inserendomi in un gruppo di persone che lavorano in squadra.

Nelle pagine che seguono, inizialmente, viene data una presentazione della Ditta ospitante, evidenziando quali sono i prodotti che essa offre e come sia organizzata al proprio interno; successivamente si passa alla presentazione dello stage, indicando il motivo che mi ha portato a scegliere un'esperienza di questo tipo e illustrando la pianificazione del Tutor Aziendale sulle attività da svolgere. Inoltre è presentato il software gestionale BluVision, alla base delle attività svolte, soffermandosi più sui dettagli di tipo informatico che su quelli più propriamente di ambito economico. Segue una descrizione del software utilizzato per la realizzazione dell'applicativo e degli strumenti utilizzati per la manutenzione e la correzione degli errori; si tratta di quegli strumenti che maggiormente sono stati utilizzati e dei quali non si avevano precedenti esperienze. In seguito vengono riportate e opportunamente spiegate le attività svolte, riguardanti alcune modifiche al gestionale per la correzione di errori segnalati e allo sviluppo di alcuni moduli aggiuntivi al gestionale; in particolare nell'ultima attività è stato sviluppato un progetto autonomamente, sfruttando le conoscenze e la padronanza degli strumenti acquisite durante lo stage, creando un piccolo

programma ad uso interno della Ditta. La relazione si conclude poi con delle considerazioni sullo stage in generale e sugli strumenti utilizzati, tenendo in considerazione quali fossero le mie aspettative e paragonandole con quello che veramente l'esperienza mi ha dato.

2. Presentazione Ditta

La ditta presso cui ho svolto il periodo di stage è la NikeZadit s.r.l. di Monselice. L'azienda è una software house che da oltre vent'anni si occupa di informatica. Nei primi anni l'attività era rivolta alla gestione totale delle problematiche informatiche, anche di tipo hardware, di aziende, istituti, enti pubblici, studi professionali e centri elaborazione dati. Dopo alcuni anni però l'esperienza e il Know How hanno reso indispensabile la specializzazione verso le piccole e medie imprese. Si è scelto quindi di centrare l'attività dell'azienda al software gestionale e ai relativi servizi di implementazione dello stesso nell'azienda.

I prodotti che NikeZadit offre sono soluzioni applicative in ambiente Windows per la gestione contabile, finanziaria, commerciale, di magazzino o di produzione per l'industria e l'azienda di distribuzione; soluzioni rivolte ad aziende che intendono impostare sul sistema informatico non solo la semplice gestione amministrativa o di magazzino ma anche in grado di controllare tutta una serie di operazioni più o meno legate all'attività aziendale che di norma non vengono gestite dai classici gestionali. Si fa riferimento per citare un esempio a quei software che gestiscono solo le operazioni di magazzino, articoli e documenti e non quegli aspetti che risultano collegati alla situazione contabile quali i cespiti, i ratei e i risconti, la contabilità analitica; servizi quindi che non è possibile gestire internamente, ma che devono essere richiesti all'esterno.

Da questo punto di vista l'azienda dedica risorse importanti allo studio e alla ricerca, migliorando costantemente la struttura organizzativa con lo scopo di garantire al cliente prodotti e servizi di qualità. L'azienda è quindi da considerarsi un produttore di software applicativo gestionale e un fornitore di servizi collegati quali Help Desk, manutenzione dei gestionali e

personalizzazioni dei prodotti su richieste specifiche dei clienti. A riguardo tuttavia deve essere fatta una precisazione. L'azienda lavora principalmente commercializzando e fornendo assistenza su un software di propria produzione, lo sviluppo del quale richiede da parte dei programmatori di rispettare certe regole sia nel modo di programmare sia nel modo di operare ed intervenire nella manutenzione dell'applicativo. Per quanto riguarda le personalizzazioni invece, queste vengono fatte da un'altra azienda, con certificazione ISO 9000 che risulta avere nome simile a NikeZadit ma che in realtà è la ditta stessa. Nel caso in cui il cliente richieda una personalizzazione del gestionale, quest'azienda, ma in realtà la stessa NikeZadit, analizza le richieste e le problematiche relative alla personalizzazione del gestionale che poi verranno comunicate alla Programmazione che cercherà di verificarne la fattibilità. Per questo tipo di operazioni l'azienda vanta una certificazione di qualità che rappresenta un importante "biglietto da visita", ma tuttavia anche un ottimo strumento di controllo e di spinta nella ricerca di soluzioni sempre migliori. Sicuramente la certificazione di qualità è uno strumento che gioca bene dal punto di vista della commercializzazione del gestionale, ma che probabilmente avrebbe garantito un prodotto migliore ed un metodo di lavoro più efficace, per quanto riguarda l'implementazione del software, se tale certificazione fosse dell'azienda che produce l'applicativo e non di quella che si occupa delle personalizzazioni.

Dal punto di vista organizzativo l'azienda risulta essere ben organizzata e vanta di personale specializzato e competente nel proprio settore e con esperienza pluriennale. Senza soffermarci sui ruoli ricoperti dai singoli dipendenti, in generale l'azienda produce al proprio interno la quasi totalità dei servizi di cui ha bisogno, occupandosi per esempio della

commercializzazione e della vendita, o della consulenza tecnica presso i clienti oltre che all'assistenza telefonica o tramite mail.

3. Presentazione Stage

La scelta di fare lo stage in NikeZadit deriva dal desiderio di mettere in pratica e migliorare le mie conoscenza informatiche cercando di integrarmi in un ambiente di lavoro organizzato e di cui non ho esperienze precedenti. Con il Tutor Aziendale si è cercato dunque di organizzare lo stage in modo tale che lo studente acquisisca le conoscenze di base del linguaggio di programmazione utilizzato per poter a fine stage metter mano autonomamente al gestionale. La pianificazione iniziale prevedeva che si arrivasse a concentrare l'attività di stage nella correzione di Bugs segnalati in fase di Test, tuttavia si è scelto poi di sviluppare qualche utilità aggiuntiva del gestionale e qualche miglioria pensata già da tempo.

Molto tempo è stato impiegato purtroppo nello studio degli strumenti utilizzati; trattandosi di un ambiente totalmente estraneo è risultato un po' difficile non tanto la comprensione del linguaggio di programmazione ma piuttosto il modo in cui è strutturato il gestionale. Diverse le fonti consultate sia cartacee sia digitali per l'acquisizione delle conoscenze base degli strumenti utilizzati, ma praticamente inesistente una guida che spieghi la struttura del progetto che è alla base del software. Di conseguenza si è vista un po' sfumare la pianificazione fatta, anche perché, piuttosto che la soluzione di Bugs come previsto, si sono poi rese indispensabili modifiche al gestionale dalla semplice modifica della grafica alla creazione di utilità ex novo da integrare nel gestionale.

Lo scopo finale dunque, indifferentemente si tratti di correzione degli errori segnalati piuttosto che di migliorie o modifiche da apportare, è quello di ottenere un buona conoscenza degli strumenti utilizzati indispensabile per poter affrontare la maggior parte degli interventi necessari alla manutenzione del software gestionale.

4. Il Software Gestionale

L'applicativo di cui abbiamo fin qui parlato è il software gestionale Bluvision; è su questo prodotto che azienda da un lato e cliente dall'altro credono e investono.

Il software gestionale Bluvision nasce su piattaforma Windows XP ma funziona con qualunque sistema operativo Windows a 32 bit.

Inizialmente l'aspetto con cui si presentava il software era stato copiato dal sistema operativo dal quale era nato, mantenendone le caratteristiche anche se sul pc il sistema operativo installato non era quello XP; per molto tempo si è scelti di mantenere questo tipo di layout, ma recentemente ci si è accorti che una soluzione migliore era quella di far adottare all'applicativo le caratteristiche del sistema operativo presente nel pc in cui viene installato. Di conseguenza ad oggi presenta icone prese dal tema XP, mentre le forms hanno uno skin diverso secondo le caratteristiche di personalizzazione del sistema operativo dell'utente.

Ricordiamo che Bluvision nasce dall'esigenza di fornire all'impresa informazioni al tempo stesso semplici e avanzate utilizzabili da tutti i componenti dell'azienda, necessarie a valutare l'andamento dei vari processi (non solo amministrativi) e a prendere con rapidità le decisioni; diverse sono dunque le aree gestionali coperte dal programma:

- L'area contabile
- L'area di controllo di gestione e di finanza
- L'area commerciale di vendita
- L'area approvvigionamenti e acquisti
- L'area Magazzino
- L'area produzione e lavorazioni esterne

Ogni operatore aziendale potrà dunque accedere a qualunque operazione e consultare gli archivi necessari; tuttavia, essendo un applicativo multi-utente, ogni operatore risulta vincolato dai permessi definiti dall'utente amministratore in fase di registrazione. La configurabilità è legata ad un sistema di restrizioni molto dettagliato che consente di definire le possibilità di visualizzazione e di gestione del programma fino al singolo campo all'interno delle forms, tanto che Bluvision è oggi uno dei pochi gestionali perfettamente adeguati alla normativa prevista dalla *legge 196 sulla Privacy e Sicurezza dei dati*. Tale normativa garantisce agli utenti che i dati personali oggetto di trattamento sono custoditi e controllati in modo da ridurre al minimo i rischi di distruzione o perdita, anche accidentale, dei dati stessi, di accesso non autorizzato o di un trattamento non consentito o non conforme alle finalità della raccolta. Inoltre il software deve adottare procedure di gestione dell'autenticazione, di aggiornamento periodico dell'individuazione dell'ambito del trattamento consentito ai singoli incaricati e addetti alla gestione o alla manutenzione degli strumenti. Un altro aspetto importante è legato all'adozione di procedure per la custodia e la protezione da trattamenti illeciti dei dati o ad accessi non consentiti. A riguardo, una delle attività di stage svolte, riguardava proprio lo sviluppo di una tecnica di cifratura per il salvataggio nel database delle password di accesso al gestionale.

Per quanto riguarda la grafica, l'interfaccia è completamente configurabile dall'utente; Bluvision consente quindi di modificare sulla base delle proprie esigenze ed abitudini operative il menù Tree View, il menù sulla barra orizzontale superiore, il menù o i vari menù configurabili sulla barra dei preferiti, le finestre contenenti gli elenchi video, le videate relative a tutte le forms, le modalità di filtro e di ordinamento delle stampe e le stampe stesse.

Con click del tasto destro su tutti gli oggetti di tutte le forms è possibile ricavare informazioni sull'oggetto stesso (nome, posizione, dimensioni). Tali informazioni possono poi essere manipolate tramite una procedura di personalizzazione del layout delle forms per utente e/o gruppo di utenti; praticamente tutti gli oggetti contenuti all'interno delle forms possono essere nascosti, disabilitati, spostati o ridimensionati. In questo modo Bluvision si adatta facilmente alle abitudini di utenti provenienti da altri programmi gestionali.

Sviluppato per ambiente Microsoft Windows, si integra perfettamente con tutti gli applicativi Microsoft Office sfruttando a pieno tutte le potenzialità; le informazioni, per esempio, derivanti da vendite di una determinata linea di prodotti possono essere caricati in un foglio Excel per essere elaborati o per stilare un resoconto da presentare ai responsabili del reparto. Ogni programma applicativo, sia Microsoft Office che altri sono richiamabili direttamente dal menù ad albero (Tree View) o dalla barra dei preferiti. Un banale esempio può essere quello delle stampe, realizzabili su diversi formati, quali Word, Pdf, Html, Txt e altri secondo le necessità del gestore. Risulta dunque una buona scelta quella del trasferimento dei dati su fogli di lavoro diversi che permettono di utilizzare le informazioni e i dati contenuti ed elaborati dal gestionale con i più comuni programmi in commercio.

È inoltre possibile collegare a molteplici archivi dell'applicazione gestionale file digitali di diverso genere, quali documenti acquisiti con lo scanner, generati da applicativi Office, generati o acquisiti da software di disegno o di grafica e acquisiti via fax; questa utilità può permettere per esempio di visualizzare l'immagine di una fattura di acquisto, di un ordine di un cliente o associare ad ogni articolo un'immagine.

Anche il servizio di posta elettronica risulta integrato al gestionale; non è necessaria la configurazione di un client di posta per utilizzare questo servizio, è possibile inviare in formato Pdf qualsiasi documento o stampa presente in Bluvision utilizzando la posta elettronica.

In genere l'applicativo viene distribuito in versione dimostrativa con limitazioni nell'uso, successivamente, una volta acquistato il prodotto, ad ogni cliente viene fornito insieme al software anche un database d'esempio che permette di far pratica sul funzionamento del prodotto senza il rischio di danneggiare il proprio database.

5. Visual FoxPro

Il linguaggio di programmazione usato è Visual FoxPro versione 9 di Microsoft, che sin dall'inizio è stato scelto per lo sviluppo del software e mantenuto negli anni a venire per l'integrazione e la manutenzione. Si tratta di un ambiente di sviluppo orientato agli oggetti per la costruzione di database e sviluppo di applicazioni, in grado di gestire ed organizzare dati per l'utente finale.
Risulta simile a C++ utilizzando una sintassi pressappoco uguale ma con una varietà di funzioni più vasta. Permette di gestire un progetto interamente e comprende tutti gli strumenti necessari per lo sviluppo di tutti i suoi moduli, dalla creazione alla compilazione e successiva esecuzione.

5.1 Struttura di un Progetto

Il "Project Manager" di Visual FoxPro è uno strumento attraverso il quale è possibile ottenere una rappresentazione grafica della struttura di un progetto; si tratta di uno strumento sicuramente molto utile che consente di organizzare tabelle, database, forms, queries, reports e altri file in fase di creazione. Può essere definito come il "centro di controllo" del lavoro che si sta svolgendo, consentendo di accedere ad ogni singolo file che costituisce il progetto, sia si tratti del singolo programma (.prg) sia di un database (.dbc) sia di una classe (.vcx).
Una scaletta molto sintetica di come viene sviluppato un progetto può essere questa:

- Definizione dei requisiti / Progetto del database
- Creazione del database, composto di tabelle e relazioni
- Creazione di un programma per l'accesso alle funzioni, completo di forms, menu, toolbar..

- Fornire all'utente l'accesso alle informazioni tramite queries (reports e grafici)
- Verifica e collaudo
- Compilazione ed esecuzione

5.2 Strumenti per lo Sviluppo di un Progetto

Ognuna di queste operazioni è facilitata da alcuni strumenti che fanno parte del programma, utili alla creazione di tabelle, forms, databases, queries e reports. Si tratta dei Visual FoxPro Designers, ognuno dei quali viene utilizzato a seconda di ciò che si vuole creare o modificare, alcuni di questi sono:

- *Form Designer* Creare una form
- *Database Desiger* Creare database e definire gli indici e le relazioni tra le tabelle
- *Query Designer* Inserire delle queries su una tabella
- *Report Designer* Creare reports (stampe) che evidenzino il risultato di una query

Ogni designers ha delle toolbars che permettono l'accesso veloce a quelle funzioni il cui utilizzo è più frequente; queste tuttavia possono essere personalizzate secondo le proprie esigenza e necessità come una qualsiasi applicazione e le modifiche verranno salvate per il successivo utilizzo.
La creazione di reports, queries e forms può essere facilitata con l'utilizzo di Wizards, una sorta di guida passo passo che aiuta il programmatore nella scelta delle impostazioni semplicemente rispondendo a delle domande e scegliendo delle opzioni.

Di particolare utilità è la Console di Visual FoxPro per mezzo della quale è possibile gestire l'apertura di tutti i files associati a Visual FoxPro. Rende possibile l'interrogazione di database e la stampa dei risultati, l'esecuzione di programmi per testarne il funzionamento e tutte le altre operazioni che in genere vengono eseguite manualmente dai menù.

5.2.1 Form Designer

Con questo strumento ricordiamo, è possibile creare una form aggiungendo campi e controlli o modificare una form esistente; tali modifiche possono essere fatte sia utilizzando il Form Wizard, sia per mezzo del Form Builder utilizzato come guida nell'aggiunta di controlli, impostando le giuste proprietà e permettendone il corretto funzionamento.
Qualsiasi caratteristica della form è gestibile e personalizzabile, rendendo possibile non solo il controllo del layout ma anche la gestione dei metodi di oggetti presenti nella form. Tra le proprietà è possibile gestire i metodi legati alla form, inserendo del codice, per mezzo del Fox Editor che vedremo in seguito, in quelli già esistenti o creandone all'occorrenza di nuovi.

5.2.2 Database / Table Designer

Il Database Designer permette sia la realizzazione da zero di un database, sia la possibilità di ottenere uno schema della struttura di un database già esistente. Aprendo un database, questo appare semplicemente come un contenitore di tabelle, delle quali vengono mostrati titolo e nome dei campi che le costituiscono, e le varie relazioni esistenti.
Per quanto riguarda le funzionalità che questo strumento offre, vediamo che è possibile visualizzare il database in diversi modi che facilitino la

comprensione della struttura o per un semplice motivo estetico; possono essere allineate in riga o in colonna oltre che per ordine alfabetico o per tipo. Nel caso più schematico invece le si può ridurre ai soli campi ed indici di interesse o al solo nome della tabella.

In fase di manutenzione si può metter mano velocemente al database o alle tabelle tramite la barra dei pulsanti del Database Designer o con il semplice click destro sugli oggetti. È possibile aggiungere tabelle o rimuoverle se queste non risultano più necessarie; deve essere rimossa anche se la si vuole utilizzare in un altro database, in quanto può appartenere ad un solo database alla volta. Una volta completato lo schema delle tabelle che devono far parte del database è possibile creare le relazioni tra le tabelle; per fare ciò è sufficiente trascinare l'indice di una tabella sull'indice della tabella a cui deve puntare, e a livello grafico otterremo delle linee tra le varie tabelle.

5.2.3 Query Designer

Una volta che è stata definita la struttura del database quello che si vuole fare è creare delle interrogazioni utili ad estrarre quei dati che soddisfano le nostre richieste. Indipendentemente da come vogliamo organizzati i dati, la strada da seguire è quella di individuare i campi all'interno delle tabelle che contengono le informazioni che cerchiamo e scegliere la destinazione a cui inviare il risultato della query.

Anche qui si può scegliere se utilizzare il Query Wizard, che creerà la query in base alle risposte date, o se utilizzare il Query Designer; si tratta appunto di procedere manualmente alla selezione della tabella o la vista contenenti i dati di interesse, definire la query scegliendo quali informazioni stampare e

l'ordine con il quale devono apparire, e che tipo di uscita utilizzare per la stampa dei risultati.

Anche qui le funzionalità sono molteplici, le informazioni delle tabelle possono essere manipolate in molti modi per estrapolare solo quella parte di informazione che cerchiamo. Alla semplice selezione dei campi di una tabella, si aggiunge la possibilità di definire un ordine di uscita e il relativo alias da usare in sostituzione del nome vero del campo; anche a livello di records è possibile intervenire, raggruppando o ordinando secondo certe restrizioni le righe da stampare. Tutte le operazioni sinteticamente descritte, non sono altro che istruzioni SQL, che è possibile visualizzare in qualsiasi momento col tasto di scelta rapida "sql" della barra dei pulsanti, ma sulle quali non è possibile intervenire.

Il risultato della query a questo punto deve essere inviato ad una delle uscite offerte; è possibile stampare le informazioni in una tabella temporanea o in una a cui viene assegnato un nome, su una finestra di browser o su quella principale di Visual FoxPro, oppure, come la maggior parte dei casi su etichette e reports.

Una volta completato il lavoro è possibile testare la query con il tasto *Run* e salvarla assegnandole un nome.

5.2.4 Report Designer

Con questo strumento è possibile creare stampe o modificare quelle esistenti. Anche qui si ripete la possibilità di utilizzare in aiuto il Report Wizard per costruire una stampa in modo facile, e utile spesso anche alla comprensione dell'utilizzo di certe funzionalità del Designer.

È possibile strutturare il report in diversi modi nella pagina, intervenendo sui controlli dei dati da stampare e sulle modalità di rappresentazione degli stessi nella stampa.
In generale un report è composto di un Intestazione, di un Piè di pagina e di una parte centrale creata a tempo di esecuzione dello stesso, operando sui records della tabella a cui è associato.
Avendo di fronte anche in questo caso una rappresentazione grafica del report, è possibile spostare e trascinare le varie parti a proprio piacimento e secondo le esigenze che si hanno. Si possono definire anche le impostazioni relative al tipo di carta utilizzato definendone margini, dimensioni e orientamento. In ogni caso, prima di procedere con la stampa si può scegliere di vedere un'anteprima del risultato ottenuto.

5.2.5 Fox Editor

Il Fox Editor non è altro che lo strumento utilizzato per la scrittura di funzioni e procedure che vengono richiamate dall'applicazione. È lo strumento maggiormente usato durante lo stage e ciò che si è potuto notare è che risulta essere ricco di funzionalità ma allo stesso tempo abbastanza facile da utilizzare. Personalizzabile come molti editor, evidenzia con colori diversi nomi di funzioni e di variabili, facilita inoltre la scrittura con l'ausilio del completamento automatico dei costrutti base del linguaggio oppure eseguendo un controllo di sintassi in automatico al salvataggio del programma segnalando eventuali errori quali un ciclo non chiuso.
Nel caso si stia operando con una procedura è possibile eseguire una ricerca di una certa variabile o funzione non solo all'interno della singola procedura, ma anche in tutte quelle appartenenti alla classe. Una particolarità infatti è che le funzioni dichiarate possono essere richiamate

ad ogni momento dell'esecuzione, come facessero parte dell'intera applicazione. A riguardo, tra i lavori di piccola manutenzione o creazione di utility nel periodo di stage, quello che sicuramente mi ha entusiasmato di più e ha richiesto maggior tempo è stato quello di creare un piccolo programma per ottenere una sorta di guida di tutte le funzioni e procedure create dai programmatori di NikeZadit, essendo queste riutilizzabili in qualsiasi momento.

5.3 Altri Strumenti utilizzati

5.3.1 Trace

Altro strumento facente parte del pacchetto Visual FoxPro, Trace permette di seguire passo passo l'esecuzione del programma, indicando la posizione all'interno del codice. Molto utile nelle operazioni di debug, consente di estrapolare i valori assunti dalle variabili, il verificarsi o meno di condizioni, di vedere tutti i cursori aperti e i loro alias, e di risalire al punto di origine di eventuali errori presenti nel codice. Per entrare nella modalità di debug basta inserire nel codice la keyword *Set Step On*; quando in fase di esecuzione Visual FoxPro incontra questa keyword apre in automatico Trace e l'esecuzione a questo punto deve procedere manualmente permettendo al programmatore di concentrarsi in certi punti. Strumento molto utile dunque e utilizzabile parallelamente al Fox Editor, per verificare l'esattezza e il corretto svolgimento del codice scritto.

5.3.2 Source Safe

Source Safe è un prodotto Microsoft, ma non appartiene a Visual FoxPro. Lo strumento permette di intervenire su di un'area di lavoro comune, apportando modifiche ai files che una volta "rilasciati" andranno a sostituire i vecchi files.
Questo strumento è utilizzato dai programmatori per modificare o correggere i files della versione più recente di Bluvision. Un file da modificare o su cui si deve lavorare, viene preso in "Check out" dal programmatore, a questo punto il file è ad uso esclusivo del singolo programmatore e agli altri compare come bloccato temporaneamente e non disponibile. Una volta completate le modifiche il file viene rilasciato e a quel punto si otterrà una versione aggiornata del file. Ad ogni intervento su di un file è possibile aggiungere un commento o una semplice spiegazione di ciò che si è fatto, per avere una storicità delle modifiche fatte al file e scovare quelle modifiche che hanno introdotto errori.

5.3.3 BluBugs

BluBugs è un programma di gestione degli errori scritto da alcuni programmatori di NikeZadit, creato ad hoc per essere utilizzato al fianco di BluVision. È ad uso interno dell'azienda e serve per segnalare, in fase di test, errori e malfunzionamenti del gestionale. Ad ogni segnalazione viene assegnato un numero progressivo, con la possibilità di indicare l'eventuale risolutore e la priorità con cui questo possibilmente deve essere risolto. Viene sempre indicato il segnalatore che deve anche lasciare una descrizione dettagliata dell'errore e la strada da seguire per rigenerarlo.
Dall'altra parte il programmatore, una volta corretto l'errore, "archivia" la segnalazione e lascia eventualmente un commento di quanto fatto.

Questi ultimi due strumenti rappresentano una buona scelta dal punto di vista organizzativo e si avvicinano molto alle prassi cui deve stare un'azienda certificata. E' facile pensare che questo tipo programmi siano stati "copiati" dall'azienda che si occupa delle personalizzazioni, che tuttavia è la stessa azienda, e che deve utilizzare questi strumenti perché vincolata dalla certificazione ISO. Risulta un po' strana a questo punto la scelta di lavorare con un software non certificato e di vantare una certificazione di qualità per le sole personalizzazioni del software stesso.

6. Sintassi del Linguaggio

6.1 Precisazioni su Visual FoxPro

Il linguaggio di programmazione utilizzato nel periodo di stage non rientra tra quelli visti durante il corso di laurea; tuttavia assomiglia molto al famoso C++, anche se Visual FoxPro presenta una gamma di funzioni e keywords maggiore.

È risultato quasi impossibile raggiungere un livello di praticità nei confronti del programma al pari dei programmatori di NikeZadit, principalmente perché il programma era nuovo, ma anche per lo scarso aiuto offerto allo studente nel primo periodo di stage nella comprensione degli strumenti usati. Tuttavia le basi di programmazione, la similarità di nomi di alcune keywords e l'Help di Visual FoxPro hanno facilitato il lavoro, permettendo allo studente di svolgere spesso i lavori autonomamente.

Visual FoxPro è un ambiente di sviluppo orientato agli oggetti che permette di creare e modificare in modo abbastanza semplice database, tabelle e oggetti grafici quali le stampe e le maschere. Risulta dunque uno strumento molto completo e adatto per la manutenzione del gestionale BluVision.

La struttura portante del gestionale è basata sulle forms, tramite le quali l'utente acquisisce dati per la stampa o si occupa della gestione aziendale inserendo dati, documenti, movimenti e informazioni che riguardano l'attività dell'azienda. Tutta l'attività dunque si riduce a delle singole forms che si interfacciano con un database fatto di un numero elevato di tabelle e un altrettanto elevato numero di relazioni tra queste.

6.2 Tipi di Dati

Non vi sono differenze rispetto agli altri linguaggi per quanta riguarda il tipo di variabili e gli operatori che è possibile utilizzare. Quelle usate, tanto per citarne alcune, sono le classiche variabili numeriche I*nteger* e *Float* o di tipo *Character* oltre alle variabili di tipo *Booleano* e *Data*; la stessa cosa vale anche per gli operatori utilizzati, siano essi numerici, logici o di confronto.

E' possibile inserire dei commenti all'interno del codice usando i caratteri "&&", da questo comando fino a fine riga le istruzioni vengono ignorate; si possono inoltre valutare alcune istruzioni e stamparne il risultato col comando "?".

Per quanto riguarda le funzioni, Visual FoxPro ne ha al suo interno moltissime, spesso inutilizzate e ignote anche ai programmatori; è capitato spesso di voler fare una certa operazione utilizzando più procedure note e poi scoprire che il tutto era fattibile da una sola funzione di cui però non si sapeva il nome. Di seguito verrà data una descrizione sintetica delle funzioni più utilizzate dallo studente per svolgere le attività e di quelle funzioni comuni a più linguaggi, che successivamente nel dettaglio dei programmi verranno spiegate.

Per quanto riguarda i cicli, tra quelli più classici ci sono:

- IF expr [THEN] com1 [ELSE com2] ENDIF
- DO CASE CASE expr1 [com1]
  [CASE expr2 [com2]] ... [OTHERWISE [comn]] ENDCASE
- DO WHILE expr com [LOOP][EXIT] ENDDO
- FOR expr com [EXIT][LOOP] ENDFOR | NEXT

Vi sono poi delle variazioni ai costrutti classici, come l'uso di IIF(expr, com1, com2) piuttosto di IF..ENDIF, o di DO WHILE..ENDDO usato al

posto di FOR..NEXT, che tuttavia risultano essere più lenti, o di funzioni create ad hoc per determinati casi come FOR EACH..ENDFOR utile all'esecuzione di una serie di comandi all'interno di un array.
È possibile inoltre manipolare stringhe ed array con diverse funzioni. Si può utilizzare ALLTRIM(expr) se si vogliono togliere da una stringa tutti gli spazi o AT se si vuole ottenere la posizione di una certa stringa di caratteri all'interno di un'altra; con SUBSTR(expr, n [,m]) si può estrarre da una stringa un numero m di caratteri a partire dall'n-sima posizione oppure utilizzando RIGHT(expr, n) e LEFT(expr, n) per ottenere i primi n caratteri partendo rispettivamente da destra e da sinistra. Tra le altre svariate funzioni, ricordiamo poi la possibilità di trasformare in maiuscolo e minuscolo (UPPER, LOWER) dei caratteri, molto utile quando si deve fare un confronto tra stringhe.
Sugli array invece, senza citare il nome delle funzioni, si può operare ottenendo informazioni sulla dimensioni e sul numero degli elementi, conoscere il numero di righe e numero di colonne, salvare le informazioni relative alla struttura dell'array in una work area, estrarre e inserire colonne e righe, copiare e cancellare elementi; tutte operazioni dunque che normalmente servono nel lavorare con gli array, alle quali se ne aggiungono altre che verranno spiegate in un secondo momento se utilizzate nel codice.

Per quanto riguarda le operazioni di interrogazione di database, quelli usati sono comandi con sintassi molto simile a quello SQL che invece è stato utilizzato durante il corso di studi.
Ci sono una serie di funzioni molto utili per spostarsi all'interno delle tabelle o ricercare alcuni record, quali: GO, LOCATE FOR, SEEK, SCAN..ENDSCAN.

Molte funzioni dunque, con svariati compiti e spesso simili ma con piccoli accorgimenti che li differenziano o semplicemente per tempi di risposta diversi. L'impressione che si ha è dunque è quella di un linguaggio di programmazione abbastanza facile e completo, tuttavia è difficile dare un giudizio obiettivo in quanto non lo si può confrontare con programmi al pari di questo.

6.3 Regole di Sintassi adottate

Sono state definite delle regole di programmazione interne all'azienda, che i programmatori sono tenuti a rispettare. Premesso che il gestionale è alla manutenzione di più persone, si è giustamente deciso di adottare un certo modo di scrivere il codice per facilitare la comprensione e velocizzarne la lettura. Nelle pagine a seguire saranno viste alcune utility sviluppate nel periodo di stage che sono state scritte, salvo qualche svista, rispettando il più possibile tali regole.

Visual FoxPro risulta molto permissivo in quanto a dichiarazione di variabili o uso di parole a metà, di conseguenza si è deciso di dichiarare esplicitamente all'inizio della funzione tutte le variabili e la relativa tipologia, anche se FoxPro le gestisce in maniera automatica; che il primo carattere del nome deve contenere il tipo di dato, oltre ad usare lettere maiuscole e minuscole per facilitarne la lettura (es. ThisForm e non thisform). È preferibile inoltre l'utilizzo di cicli standard, senza ricorrere alle variazioni che offre FoxPro, l'uso di identazione e il rispetto dell'ordine della sintassi nello scrivere le istruzioni senza l'abbreviazione dei comandi. Per le funzioni create dovrà essere indicata la spiegazione di cosa esegue e dei dati che necessita in input e di quelli che restituirà; in particolare il commento deve essere del tipo **!** per essere distinguibile.

È consigliato di spezzare funzioni troppo lunghe, creando funzioni più piccole che svolgano singole operazioni e che possano essere riutilizzabili in altre occasioni, evitando così la duplicazione del codice. Utilizzare poi le classi e i controlli esistenti in modo corretto, prendendo per esempio *FrmInput* se si deve realizzare delle form che eseguano operazioni e non altre tipologie di classi. Ultima nota per i commenti, che devono essere inseriti all'interno del codice per facilitarne l'interpretazione o per spiegare alcune scelte.

Tutta questa serie di "regole" di scrittura sono un ottimo modo di implementare il software e rimandano al discorso della certificazione di qualità. Non è possibile stabilire se l'utilizzo di queste regole sia una scelta dell'azienda o se derivi dall'adottare gli stessi meccanismi che regolano l'azienda certificata, sta di fatto che sicuramente è un qualcosa di vantaggioso ed in linea con quelle regole che un'azienda ben strutturata dovrebbe avere.

7. Modifiche e Utility per BluVision

Come anticipato, l'intenzione del Tutor Aziendale era quella di far correggere allo studente gli errori segnalati, inizialmente affiancato da uno dei programmatori con l'intenzione di arrivare ad una certa autonomia nella scelta delle modifiche da effettuare. I primi interventi al gestionale infatti riguardavano delle correzioni a tabelle e ad un report che non si sono volute fare intervenendo manualmente ma scrivendo dei programmi. Questa scelta si è rilevata molto utile; non solo a livello pratico, perché semplificava di non poco il lavoro, ma anche perché è servito a capire qualcosa in più del progetto. Una volta individuato dove intervenire per effettuare le modifiche, non si sono avute grosse difficoltà nello scrivere le istruzioni, tuttavia il tempo impiegato è giustificato dal fatto che la struttura alla base del gestionale non era chiara. Successivamente dalle modifiche si è passati alla creazione di due forms che gestissero un malfunzionamento del gestionale la cui spiegazione verrà data in seguito. Un ultima cosa fatta poi è relativa alle funzioni di Visual FoxPro scritte dai programmatori di NikeZadit che si è protratta fino alla fine dello stage.

7.1 *AlterFields.prg*

Su segnalazione di un bugs, ci si è accorti che salvando su alcune tabelle del database `Dblocal` le informazioni relative a data e ora inserite su certe forms, non venivano successivamente restituite in modo corretto. Da una semplice analisi su alcune delle tabelle in causa, è stato possibile verificare che il problema risiedeva nel tipo di dati che teoricamente ci si sarebbe aspettato in alcuni campi; quello che si doveva fare era dunque cambiare il data type di alcuni fields da *date* a *datetime*. Poiché si dovevano passare al setaccio un numero imprecisato di tabelle e per maggiore sicurezza si è deciso di scrivere un programma che controllasse ogni singolo campo di ogni tabella alla ricerca degli errori.

Vista l'occasione, se ne è approfittato per effettuare un'altra correzione necessaria prevista da tempo ma non ancora attuata; è stata modificata la lunghezza di due campi di tipo carattere aumentandola da 20 a 40.

Qui sotto è riportato il programma scritto:

```
* cambia il tipo dei campi timeins e timemod da D a T;
* imposta la lunghezza dei campi userins e usermod a 40 e
* stampa la lista dei nomi modificati in un file di nome
  alterfields.txt

LOCAL Dblocal, m, n, i, z, cField, t

Dblocal=GETFILE('dbc')
OPEN DATABASE FULLPATH(Dblocal)
CD path
m = ADBOBJECTS(aDblocal,"TABLE")
DELETE FILE("alterfields.txt")
hfile = FCREATE("alterfields.txt")
FPUTS(hfile,"TABELLE MODIFICATE")
t=0
FOR i=1 TO m *scorro tutte le tabelle
  USE (aDblocal(i)) IN 0 ALIAS tbl EXCLUSIVE
  SELECT tbl
  n = AFIELDS(aDb)
  CLEAR
  FOR z = 1 TO n *scorro tutti i fields
```

```
    IF UPPER(aDb(z,1))= "TIMEINS" AND TYPE(aDb(z,1))!= "T"
      t=1
      cField="TIMEINS"
      FPUTS(hfile,"-" + aDblocal(i) + ":
                    modificato campo " + cField)
      ALTER TABLE aDblocal(i) ALTER timeins T
    ENDIF
    IF UPPER(aDb(z,1))= "TIMEMOD" AND TYPE(aDb(z,1))!= "T"
      t=1
      cField="TIMEMOD"
      FPUTS(hfile,"-" + aDblocal(i) + ":
                    modificato campo " + cField)
      ALTER TABLE aDblocal(i) ALTER timemod T
    ENDIF
    IF UPPER(aDb(z,1))="USERINS"
      ALTER TABLE aDblocal(i) ALTER userins c(40)
    ENDIF
    IF UPPER(aDb(z,1))="USERMOD"
      ALTER TABLE aDblocal(i) ALTER usermod c(40)
    ENDIF
  ENDFOR
  USE
ENDFOR

IF t=0
FPUTS(hfile,"Nessuna tabella modificata.")
ENDIF

FCLOSE(hfile)
```

È ordinata l'apertura di una finestra di dialogo per cercare il database di interesse e, una volta individuato, con la funzione *ADBOBJECT()* viene creato un array con i nomi delle "*TABLE*" presenti. Successivamente con un ciclo FOR si entra in ogni singola tabella e per ognuna viene creato di volta in volta un array *aDb* contenente tutte le informazioni che la riguardano; tra queste quelle nella prima colonna contengono i fields name. Un altro ciclo *FOR* dunque viene lanciato alla ricerca dei campi da modificare; al programma è inoltre richiesto di creare un file *alterfields.txt* contenente nome delle tabelle e dei campi modificati.

7.2 *MdfFiltro.prg*

Altro tipo di correzione su segnalazione di bugs è quella relativa al layout di una stampa. Un report, giustamente, riporta ad inizio della pagina stampata una sintesi contenente le informazioni sui dati stampati e sui relativi filtri impostati dall'utente che lo hanno generato. Il problema riscontrato in questo caso era che le informazioni venivano introdotte in ogni pagina del report, con relativa importanza se queste prendevano poche righe, ma un problema invece se occupavano metà della pagina. Per ovvi motivi di layout e spreco inutile di carta, si è pensato di inserire le informazioni limitatamente alla prima pagina e non in tutte quelle consecutive.
Anche in questo caso si è intervenuti scrivendo poche righe di codice che facessero il lavoro.

```
* MdfFiltro.prg
* modifica il contenuto del campo print when per stampare il
  filtro solamente sulla prima pagina

LOCAL pathStmp, nC, nF, n, m

pathStmp=FULLPATH(GETDIR())

CD (pathStmp)

nC = ADIR(aCart,"*.","D")

FOR n = 3 TO nC
  CD FULLPATH(pathStmp+(aCart(n,1)))
  nF = ADIR(aFile,"*.frx")
  FOR m = 1 TO nF
    USE aFile(m,1) ALIAS file
    SELECT file
    SCAN FOR UPPER(expr)='OREPORT.CDESFILTRO'
      REPLACE Supexpr WITH ALLTRIM(supexpr) + "
                           AND _PAGENO=1"
    ENDSCAN
    USE
  ENDFOR
ENDFOR
```

I report su cui bisognava apportare le correzioni erano sparsi in più cartelle di un'unica directory, dunque si è inizialmente cambiata la directory di default di FoxPro settandola al percorso della cartella di interesse, poi si è creato un array *aCart* contente i nomi delle sotto directory presenti al suo interno e di ogni cartella si è creato un array *aFile* con i file di tipo *frx* (report) presenti all'interno. Una particolarità di FoxPro è che i reports possono essere trattati come delle tabelle, dunque vengono aperti e modificati con i comandi SQL classici. Noto che, i fields da modificare avevano in comune una certa stringa nel campo *expr*, al programma viene richiesto di considerare solo i record con questa caratteristica e dunque di modificare il corrispondente field *Supexpr* aggiungendo l'istruzione "*AND _PAGENO=1*" con il comando di modifica records *REPLACE*.

Il problema di correggere la stampa dunque si risolveva nell'aggiunta di due istruzioni, ma anche in questo caso i file da modificare non erano ben noti e un lavoro di modifica manuale non avrebbe garantito l'inserimento delle istruzioni corrette in tutti i reports.

7.3 *CriptPsw e CriptPswTbl*

L'inserimento di un nuovo utente in Bluvision è gestito dall'utente amministratore. Viene creato un nuovo profilo mentre la password viene lasciata vuota e da cambiare al primo accesso.
Inizialmente le password venivano semplicemente salvate in una tabella contenente tutte le informazioni e i permessi di un singolo utente; tabella tuttavia visionabile dal solo utente admnistrator. Per rendere più sicuro l'accesso e per la privacy dei singoli utenti prevista dalle normative della legge 196 sulla Privacy e Sicurezza dei dati, si è deciso di sviluppare una funzione per criptare le password e di inserirla successivamente all'interno del programma che gestisce l'accesso a BluVision.
Il meccanismo scelto per criptare è quello di trasformare in codice ASCII i caratteri che compongono la password e una volta sommato un certo valore ad ognuno di essi, trasformarli nuovamente in carattere; si ottiene così una sigla di caratteri che non possono essere decifrati se non da chi è a conoscenza della procedura che li ha prodotti. Tuttavia, il meccanismo adottato risulta un po' banale se si pensa alla riservatezza e all'importanza che le informazioni memorizzate nei database non vengano manomesse o distrutte. Un uso improprio o illecito delle funzioni del gestionale o la manipolazione da parte di persone poco esperte potrebbe portare a conseguenze rilevanti per l'azienda. Questa scelta dunque sembra discutibile se associata agli aspetti appena visti, anche se tuttavia in fase di progettazione tali presupposti non sono stati considerati dalla Programmazione.

Una volta studiata la soluzione migliore per gestire i parametri in ingresso e quelli in uscita, la funzione si è risolta nelle poche righe di codice riportate di seguito.

```
FUNCTION CriptPsw (cPsw)
  ** CriptPsw: Ritorna la password criptata
  ** Parametri in ingresso: - cPsw: password
  ** Parametri in uscita: - cPswCript: password criptata

  LOCAL nLenPsw, i as Integer
  LOCAL cPswCript as String

  cPswCript = ''
  nLenPsw = LEN(cPsw)

  FOR i = 1 TO nLenPsw
    cTemp = SUBSTR(cPsw,i,1)
    cTemp = ASC(cTemp) + 33
    cTemp = CHR(cTemp)
    cPswCript = cPswCript + cTemp
  ENDFOR

  RETURN cPswCript
ENDFUNC
```

La funzione riceve in input la variabile cPsw che inizialmente invece veniva copiata direttamente sul database; dopo essere stata criptata (cPswCript) viene ritornata al programma che procede con il salvataggio. Una piccola precisazione a riguardo è dovuta per precisarne il funzionamento. Al primo accesso è richiesto all'utente di inserire una propria password per proteggere il proprio profilo. La form che salva la password in verità non fa altro che sostituire la password vuota fornita per il primo accesso con quella nuova. Si tratta dunque di un "cambio password" nel database gestito al primo accesso a BluVision. La funzione tuttavia non viene utilizzata soltanto per l'inserimento di una nuova password, ma anche ogni volta che è richiesta la verifica della password inserita; questa infatti per coincidere

con la stringa di caratteri presente nel database deve essere a sua volta criptata e confrontata, se il risultato è "TRUE" allora è consentito l'accesso. Visto il metodo di criptaggio adottato, si distinguono dunque caratteri in maiuscolo da quelli in minuscolo, proprio perché la loro corrispondenza in codice ASCII non è uguale.

Il problema a questo punto è risolto per tutti i nuovi utenti che vengono inseriti, ma non lo è per quelli già presenti; ai vecchi utenti che volessero entrare col proprio user al gestionale, sarebbe negato l'accesso, in quanto la password inserita una volta criptata non risulterebbe uguale a quella salvata nel database; ricordiamo infatti che per tutti quei profili registrati prima che venisse gestito il salvataggio delle password, queste risultavano salvate normalmente.

Di conseguenza si è reso necessario anche un intervento sui campi della tabella contenente le password per garantire l'accesso agli utenti con profilo già registrato.

Di seguito è mostrato il programmino creato, lanciato dall'interno del gestionale con il compito di entrare nei campi della tabella e criptare le password.

```
&& CriptPswTbl.prg

LOCAL nLenPsw, i, z as Integer
LOCAL cPswCript, cDbsys as String

CLOSE DATABASES

cDBsys=GETFILE('dbc')  && Ricerca del Database
OPEN DATABASE FULLPATH(cDBSYS)
USE utenti IN 0 EXCLUSIVE

SELECT utenti
SCAN
  IF Utenti.PswCripted=.F.
    IF !EMPTY(Utenti.Password)
      cOldPsw = ALLTRIM(LOWER(Utenti.Password))
      cNewPsw = CriptPsw(cOldPsw)
```

```
      REPLACE password WITH cNewPsw
      REPLACE PswCripted WITH .T.
    ENDIF
    IF !EMPTY(utenti.pswold1)
      cOldPsw1 = ALLTRIM(lower(utenti.pswold1))
      cNewPsw1 = CriptPsw(cOldPsw1)
      REPLACE pswold1 WITH cNewPsw1
    ENDIF
    IF !EMPTY(utenti.pswold2)
      cOldPsw2 = ALLTRIM(LOWER(utenti.pswold2))
      cNewPsw2 = CriptPsw(cOldPsw2)
      REPLACE pswold2 WITH cNewPsw2
    ENDIF
  ENDIF
ENDSCAN

USE
CLOSE DATABASES


PROCEDURE CriptPsw
     && stesso meccanismo di criptaggio utilizzato per
     && gestire l'accesso

  LPARAMETERS cPsw
  cPswCript = ''
  nLenPsw = LEN(cPsw)

  FOR i = 1 TO nLenPsw
      cTemp = SUBSTR(cPsw,i,1)
      cTemp = ASC(cTemp) + 33
      cTemp = CHR(cTemp)
      cPswCript = cPswCript + cTemp
  ENDFOR

RETURN cPswCript
ENDPROC
```

Alcune condizioni sono state inserite però per mirare soltanto ai campi d'interesse. La prima è che la password non sia già stata criptata, controllo da effettuare sul campo `Utenti.PswCripted`, e poi che il campo non sia vuoto. Infatti, mentre si testava il programma si è reso necessario

intervenire manualmente sulla tabella ed inserire il campo `PswCripted` per controllare che si criptassero le password “vecchie”.

All’interno del programma come vediamo è richiamata la procedura `CriptPsw` che ovviamente deve criptare le password seguendo le stesse operazioni della funzione sopra.

Altro cosa da considerare e di grossa importanza che si è dovuta gestire è che i “vecchi utenti” entravano nel proprio profilo scrivendo la password indifferentemente con caratteri maiuscoli o minuscoli e soprattutto molti non ricordavano come questa fosse stata inserita inizialmente. Il problema è stato risolto trasformando la password in caratteri minuscoli (`LOWER(Utenti.Password)`)e successivamente criptata; sono stati avvisati poi tutti gli utenti delle nuove modifiche apportate e che la password da inserire per entrare nel proprio profilo doveva necessariamente essere in minuscolo; per chi poi volesse cambiarla, di fare attenzione a come venisse scritta, trattandosi a questo punto di una verifica case-sensitive della password inserita.

7.4 *I_tsk.scx*

Risolto il problema password, si è deciso di accantonare la risoluzione dei bugs per affrontare invece un problema che continuava a presentarsi utilizzando BluVision. Se pur saltuariamente, succedeva che alcune forms aperte uscivano dallo schermo e non era possibile recuperarle o che addirittura si bloccassero mentre le si utilizzava; non essendo possibile chiuderle o riaprirle, l'unica soluzione era quella di uscire dal programma ed interrompere il proprio lavoro, cosa poco gradevole per un gestionale del calibro di BluVision. Non riuscendo a capire da cosa derivasse il problema, la soluzione migliore al momento era di creare una sorta di "Task Manager" come quello di Windows che chiudesse le forms aperte; una form dunque che contenga l'elenco delle forms di FoxPro in uso in quel momento e la possibilità di chiuderle all'occorrenza.

La classe da utilizzare in questo caso è la *FrmInput* di Visual FoxPro personalizzata dai programmatori per BluVision, che presenta già le caratteristiche necessarie, di conseguenza nel disegnarla ci si è limitati a posizionare gli oggetti necessari: i bottoni di conferma e di uscita e un field dove stampare l'elenco delle forms aperte. Il lavoro da fare stava nel modificare alcuni dei metodi scrivendo del codice per far eseguire alla form le operazioni desiderate. Tra gli strumenti offerti da FoxPro ricordiamo, c'è una finestra in cui sono presenti tutti i controlli che si possono gestire costruendo una form, relativi a layout, metodi e quant'altro. Nella lista dei metodi in particolare sono compresi anche quelli creati su misura dai programmatori, nel nostro caso tre dei quattro usati hanno questa particolarità. Si tratta dei metodi *init, clicktb, okfrm* e *privatetb* che ora vedremo singolarmente.

PrivateTb e *ClickTb* sono due metodi creati e personalizzati dai programmatori per BluVision. Sono due metodi definiti a livello di classe e riportati uguali in ognuna di esse. Più precisamente PrivateTB è quello che

serve ad aggiungere i pulsanti nella form e come vediamo di seguito contiene poche istruzioni .

```
This.CntTB.AddObject("BtnClose", "ButtonTB",
  "Images\Bmp\EsciTB.Bmp", "Annulla / Chiudi", This.CntTB)
This.CntTB.AddObject("BtnOK", "ButtonTB",
  "Images\Bmp\SalvaTB.Bmp", "Chiudi Form", This.CntTB)
DODEFAULT()
```

Il *dodefault()* fa eseguire il codice della classe padre *FrmInput* e poi quello della classe superiore ancora *FrmBase,* permettendo di aggiungere i pulsanti standard già definiti nella classe. Possono poi essere aggiunti altri pulsanti nella singola form con il metodo *AddObject,* nel nostro caso sono stati aggiunti i pulsanti “Annulla” e “Chiudi Form” al container di oggetti *CntTB* che deriva dalla classe personalizzata *ButtonTB*. Le informazioni poi da passare al metodo sono relative all’immagine a inserire come pulsante e alla tool tip text, cioè il testo che verrà visualizzato quando ci si posiziona col mouse sul pulsante.

Per quanta riguarda *ClickTb* possiamo dire che principalmente è presente nella classe delle forms, ma in molti casi viene passato anche alle varie form. Questo metodo serve a gestire la pressione del pulsante, in particolare di BtnOk, eseguendo le istruzioni di OkFrm se sono soddisfatte una serie di condizioni, altrimenti se questo non viene premuto viene eseguito il codice della classe *dodefault.*

```
LPARAMETERS cButton

IF UPPER(cButton) == "BTNOK" THEN
  IF Innesto("Inn" + This.Name, PreOk, This) THEN
    IF This.ChkDataFields() THEN
      lValidFrm = This.ValidFrm()
      IF (TYPE("lValidFrm") = "L" AND lValidFrm) OR
      (TYPE("lValidFrm") = "N" AND lValidFrm > 0) THEN
        This.IsOkFrm = .T.
        This.OkFrm()
```

```
        Innesto("Inn" + This.Name, PostOk, This)
      ENDIF
    ENDIF
  ENDIF
ELSE
  DODEFAULT(cButton)
ENDIF
```

Init è l'unico metodo utilizzato che appartiene a quelli base di Visual FoxPro ed è quello che viene scatenato aprendo la form. Nel nostro caso dunque le istruzioni che devono essere inserite saranno di creare un elenco delle form utilizzate in quel momento.

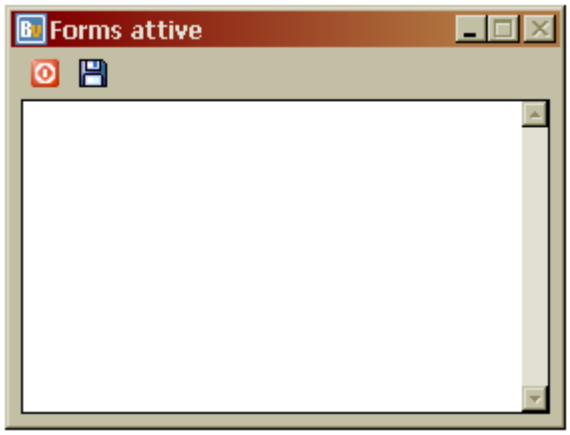
```
LPARAMETERS lLinked, lNoShow, lFrmUser, lModal, lNoInnesto
LOCAL oForm as Object
LOCAL n, nForms as Integer
LOCAL bOut as Boolean
bOut = DODEFAULT(lLinked, lNoShow, lFrmUser, lModal,
lNoInnesto)

IF bOut THEN
  nForms = 0
  ** Faccio un ciclo per le form aperte
  FOR n = 1 TO _SCREEN.FormCount
    oForm = _SCREEN.Forms(n)
    ** Se è la TB o la 3View o la barra dei preferiti...
    IF UPPER(oForm.BaseClass) != "TOOLBAR" AND
    !INLIST(UPPER(oForm.Class), "MAIN3VIEW", "PREFERITI")
    AND oForm.ShowWindow < 2 AND oForm.Name != This.Name
    THEN
      nForms = nForms + 1
      ThisForm.LstBaseLbl1.Field.AddItem(oForm.caption)
      ThisForm.LstBaseLbl1.Field.List(nForms,2)=oForm.name
    ENDIF
  NEXT
ENDIF
RETURN bOut
```

La videata principale di Visual FoxPro è trattata in questo caso come fosse un oggetto e con la proprietà *FormCount* si riesce a contare il numero di forms usate in quel momento; successivamente con *Forms()* viene creato in

run time un array che permette l'accesso in ognuna di esse. Per ognuna vengono dunque verificate alcune condizioni per l'inserimento nell'elenco che si vuole creare: sono escluse quelle la cui classe risulta essere di tipo *main3view* o *preferiti*, perché appartenenti alla schermata principale di BluVision, e quelle di tipo *toolbar*; ShowWindow serve per escludere invece quelle form di tipo top-level, nelle quali normalmente sono inserite altre forms; è ovviamente esclusa la form che si sta creando.

Il tipo di field usato per l'elenco ha una particolarità, permette di stampare la *caption* della form e di tenere a memoria in un'array il nome corrispondente, viene dunque usata la proprietà *AddItem()* per aggiungere la voce all'elenco, mentre si utilizza *List()* per creare un'ulteriore colonna che tuttavia non viene mostrata, e dove viene inserita nella corrispondente riga *nForms* il nome vero. È stato fatto questo perché la caption contiene il nome che compare in testa alla form, mentre la proprietà *name* restituisce il nome con cui è salvata, quello che ci serve per poterne ordinare il rilascio. Poiché l'utente riconosce la form basandosi sulla caption e non sul suo "vero" nome, per non creare equivoci si è reso necessario procedere in questo modo; meccanismo un po' più laborioso ma con la soluzione migliore pensando all'utilizzatore finale. La form pronta all'utilizzo si presenta in questo modo:



Un ultimo metodo da modificare perché l'utility possa funzionare, è quello che gestisce il rilascio della form che si vuole chiudere. Le istruzioni vengono dunque inserite nel metodo *okfrm* che viene scatenato premendo il tasto *Chiudi Form*. Si esce semplicemente invece scegliendo il tasto *Annulla / Chiudi*.

```
LOCAL oForm as Object
LOCAL n, nChiusForm as Integer

FOR n = 1 TO _SCREEN.FormCount
  oForm = _SCREEN.Forms(n)
  IF oForm.Name ==
  ThisForm.LstBaseLbl1.Field.List(ThisForm.LstBase
  Lbl1.Field.ListIndex,2) THEN
    nChiusForm = BluMsgBox("Voui chiudere
    '"+oForm.Caption+"'?",4+32,"Conferma chiusura Form")
    IF nChiusForm = 6
    *controllo form di input aperte
      IF UPPER(oForm.Class) == 'FRMELENCO' AND
      WEXIST(oForm.cFrmInput) THEN
        BluMsgBox("Chiudere prima la form di input")
        EXIT
      ELSE
        * controllo form modale aperte
          IF _SCREEN.Forms(n-1).WindowType = 1
              _SCREEN.Forms(n-1).Release()
          ENDIF
        oForm.Release()
        ThisForm.lstbaselbl1.field.RemoveItem(ThisForm.Lst
        BaseLbl1.Field.ListIndex)
        ThisForm.Release()
        _SCREEN.Refresh
      ENDIF
    ENDIF
  ENDIF
NEXT
oForm = .F.
```

Brevemente, quello che dovrebbe fare la procedura è di ricavare il nome della form selezionata e di ordinarne il rilascio. La soluzione inizialmente prevedeva un ciclo per le form aperte alla ricerca di quella candidata ad

essere chiusa e all'OK del messaggio di conferma, questa veniva rilasciata. Completata l'utility però, durante i test per verificarne il corretto funzionamento, sono emersi alcuni problemi. È facile immaginare che, utilizzando il gestionale, le forms che vengono aperte in simultanea possono essere numerose e come spesso capita da una form ne viene aperta un'altra, per l'aggiunta ad esempio di un nuovo cliente nella lista clienti, piuttosto che per la selezione di un diverso filtro per la selezione dei dati. Tutte le varie possibilità sono comunque riconducibili a due casi controllabili: il primo è relativo a quelle form che permettono l'inserimento di nuovi dati e la cui classe deve appartenere necessariamente alla classe *FrmInput*, dunque la verifica da effettuare è controllare se la classe di appartenenza della form da chiudere è di tipo elenco e se esiste una form di input aperta da questa; l'altra possibilità riguarda invece le forms che per un qualsiasi motivo ordinano l'apertura di un'altra form ma in modalità *modal*, cioè aprono una form solo sulla quale è consentito operare disabilitando temporaneamente l'accesso alle altre forms aperte o per lo meno finché questa non viene rilasciata. In questo caso non viene lanciato un messaggio con la richiesta di chiudere la form come per il primo caso, ma viene in automatico verificata l'apertura di una form modal ed eventualmente rilasciata. Quindi si procede togliendo la form dall'elenco, chiudendo poi anche l'utility stessa e aggiornando lo *_SCREEN*.

7.5 *I_reslay.scx*

La creazione della form precedente, ha dato lo spunto per la realizzazione di un utility per il ripristino delle impostazioni iniziali delle singole form. Ricordiamo infatti che BluVision apre le forms con le stesse dimensioni e posizione nello schermo impostate dall'utente all'ultimo accesso; tiene dunque a memoria le informazioni relative alla personalizzazione delle singole form e le ripropone di volta in volta.

Per realizzare questa utility, si è pensato di creare una form che elenchi tutte le forms appartenenti a BluVision e consenta la selezione di quelle a cui si vogliono cancellare le personalizzazioni fatte ripristinando quelle iniziali. Anche in questo caso la classe di appartenenza è *FrmInput* e i metodi da modificare sono gli stessi di I_TSK.

Invariato rimane il codice all'interno dei metodi *ClickTB* e *PrivateTB* che come abbiamo visto servono a far ereditare alla form le caratteristiche della classe di appartenenza.

Vediamo dunque quali istruzioni sono state inserite all'interno del metodo *Init* in questo caso, che ricordiamo è il metodo scatenato all'apertura della form.

```
LPARAMETERS lLinked, lNoShow, lFrmUser, lModal, lNoInnesto
LOCAL oForm as Object
LOCAL nForms, m, nNumLns, n as Integer
LOCAL bOut as Boolean
LOCAL cFrmDscr, cFrmName, cDefaultPath, cPath as Character

bOut = DODEFAULT(lLinked, lNoShow, lFrmUser, lModal,
lNoInnesto)

ThisForm.BorderStyle = 3
*Sizable..
cDefaultPath = CURDIR()
*cPath = oApp.AppDir + "BIN\FORMS\"
*SET PATH TO &cPath
CD BIN\FORMS
nForms = ADIR(aForms,"*.scx")
```

```
oApp.MousePointer = 11
oApp.oTV.oTV.MousePointer = 11

CREATE CURSOR FormCurs (Sel Logical, Descr Character(100),
FormName Character(20))
* Creo i tag con lo stesso nome del campo da ordinare
INDEX ON Descr TAG Descr
INDEX ON FormName TAG FormName

FOR m = 1 TO nForms
  IF aForms(m,1) != "I_ResLay.scx"
    USE aForms(m,1) ALIAS oForm IN 0 SHARED AGAIN
    SELECT oForm
    SCAN FOR UPPER(SUBSTR(oForm.Class,1,3)) = "FRM"
      cFrmDscr = MLINE(oForm.Properties,ATCLINE('caption =
      "', oForm.Properties))
      cFrmDscr = STREXTRACT(cFrmDscr,'"','"')
      cFrmName = MLINE(oForm.Properties,ATCLINE('name = "',
      oForm.Properties))
      cFrmName = STREXTRACT(cFrmName,'"','"')
    ENDSCAN
    SELECT FormCurs
    INSERT INTO FormCurs (Sel, Descr, FormName)VALUES (.F.,
    cFrmDscr, UPPER(cFrmName))
    USE IN oForm
  ENDIF
NEXT

ThisForm.GrdTeste.RecordSource = "FormCurs"
ThisForm.GrdTeste.ColSel.ControlSource  = "Sel"
ThisForm.GrdTeste.Column2.ControlSource  = "Descr"
ThisForm.GrdTeste.Column3.ControlSource  = "FormName"
GO TOP
CD ..\..

oApp.MousePointer = 1
oApp.oTV.oTV.MousePointer = 1
```

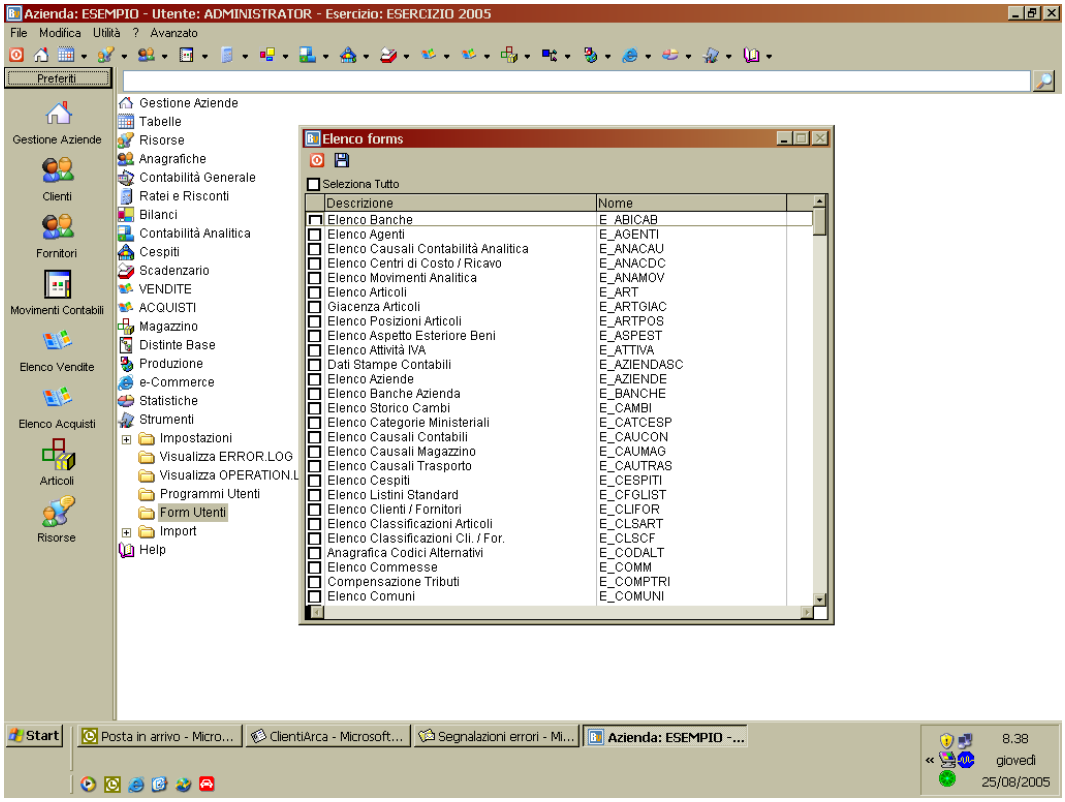
Vengono eseguite le normali procedure per la selezione della cartella di interesse e costruito l'array *aForms* contenente tutte le informazioni delle form presenti.

In questo caso però le informazioni di interesse non vengono inserite in un campo *txt* di elenco qualunque, ma in una *GrdTeste*; si è scelto di usare un

field di questo tipo perché può essere popolato da una tabella o da un cursore contenenti le informazioni. Un cursore infatti è stata la scelta adottata, inserendovi temporanea le informazioni per popolare l'elenco. Le informazioni da cercare e da inserire nella *Grid* riguardavano la caption della form, per facilitare la ricerca, e il nome con cui questa era stata salvata, inoltre è stata aggiunta una colonna di selezione per dare la possibilità all'utente di selezionare più form all'interno dell'elenco. Tre risultano dunque alla fine le colonne della *Grid* e tre i campi del cursore: *Sel*, *Descr* e *FormName*.

Per inserire i record all'interno del cursore, col ciclo FOR si sono aperte le varie form trattandole come fossero tabelle, lo stesso procedimento ricordiamo usato per la modifica del filtro di un report. Ogni form aperta appare dunque con dei fields di tipo *memo* con all'interno tutte le proprietà che la caratterizzano. Verificata l'appartenenza a classi che iniziassero con i caratteri "*frm*" e noto che le informazioni di interesse erano contenute nel field *Properties*, con le funzioni *MLINE* e *ATCLINE* sono state copiate le linee contenenti la caption ed il name in due variabili. A questo punto le informazioni raccolte vengono inserite nel cursore e impostato a false il campo di selezione.

Completato il ciclo per tutte le forms, i records del cursore vengono copiati nella Grid e si ottiene la form come mostrato nella pagina seguito.



Poiché l'operazione in esecuzione richiedeva qualche secondo, è stato cambiato il puntatore del mouse assegnandogli l'immagine della clessidra per indicare l'attesa della fine delle operazioni.

Sono stati aggiunti poi rispettivamente nell'*header.click* rispettivamente della seconda e terza colonna le seguenti istruzioni che permettono l'ordinamento alfabetico delle voci presenti a seconda del tag inserito.

```
HeaderClick(This.Parent.Parent,"FormName")
HeaderClick(This.Parent.Parent,"Descr")
```

Molto brevi le istruzioni da inserire nell'*okfrm*; viene utilizzata una funzione creata dai programmatori appositamente per cancellare le nuove impostazioni, che si chiama *DelKeyRegistry* alla quale vengono passati

come parametri il *path* dove intervenire e il nome della form. Come è possibile notare, il percorso che viene passato è composto da una parte fissa e una parte che deve essere ricavata in run time; ricordiamo che Bluvision è un applicativo multi azienda e multi utente, necessita dunque delle informazioni, relative all'utente che sta usando il gestionale e in quale azienda sta operando, per cancellare la giusta cartella. È sufficiente dunque cancellare la cartella con nome uguale a quello della form per ripristinare le impostazioni di default di BluVision.

```
LOCAL cPath as Character
LOCAL nConf as Integer

cPath = "SOFTWARE\NZR\VISION\USERPROFILE\" + oApp.UserName +
"\" + oApp.Azienda + "\"

nConf = BluMsgBox("Ripristinare le impostazioni
iniziali?",4+48,"Richiesta conferma")

IF nConf = 6
  SCAN FOR FormCurs.Sel = .T.
    DelkeyRegistry(cPath, ALLTRIM(FormCurs.FormName))
  ENDSCAN
ENDIF

ThisForm.Release()
```

Viene fatto uno *SCAN* di tutte le forms selezionate cancellandone le impostazioni personali e alla fine viene ordinata la chiusura dell'utility.

7.6 *Progetto Guide Maker*

L'ultimo lavoro svolto in NikeZadit è sicuramente quello che ha dato maggior soddisfazione rispetto alle utility svolte in precedenza. È stato creato infatti un piccolo progetto gestendolo quasi autonomamente dall'inizio alla fine e vedendolo effettivamente funzionare.

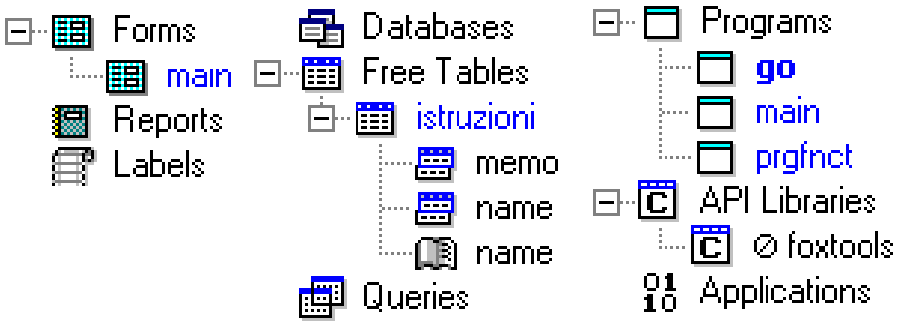
Il presupposto di questo lavoro era di creare una guida interna all'azienda che riportasse nel dettaglio tutte le funzioni e procedure create dai programmatori all'interno del progetto BluVision. Nel corso degli anni sono state molte le funzioni create e spesso capita di non ricordare nemmeno che tipo di operazioni svolga una certa funzione o quale sia il valore che restituisce; all'interno del progetto risulta infatti abbastanza difficile muoversi se non si è abbastanza esperti della struttura o per chi, come i tecnici, non è sempre a diretto contatto con i programmatori. Si richiedeva dunque un "help" relativo alle funzioni scritte che fosse d'aiuto sia ai programmatori sia ai tecnici, ma visionabile in caso di bisogno anche da tutti i dipendenti dell'azienda.

I programmatori erano già in possesso di un programma che ricava tutte le informazioni relative a classi, funzioni, database e menu da un progetto, ma questo non faceva altro che entrare nella struttura e copiare quello che trovava presentandolo poi con un layout poco comprensibile. Non era di certo quello che si cercava, ma per alcuni giorni si è provato a metter mano a questa utility senza ottenere grossi risultati; si è deciso dunque di abbandonare il programma e di creare un nuovo progetto.

Il nuovo progetto ha subito diverse modifiche in corso di creazione, soprattutto per migliorare il layout delle pagine di output, rivedendo e modificando di volta in volta il codice scritto o addirittura riscrivendo tutto per seguire strade con soluzioni migliori e di più facile lettura. Nello

scrivere si è cercato di rispettare il più possibile le regole di sintassi, che brevemente sono riportate sopra, dichiarando le variabili e assegnando nomi adeguati e mantenendo un certo ordine utilizzando l'identazione, inserendo qualche commento e rimandando certe operazioni a funzioni più piccole. Per quanto riguarda il tipo di output, si è scelto di inserire le informazioni in pagine *html,* integrando così due diversi linguaggi di programmazione e permettendo di sfruttare le conoscenze acquisite in merito durante il corso di laurea.

La struttura del progetto è abbastanza semplice, è composta da una form *main*, da una tabella *istruzioni* e da tre programmi. Qui sotto ne viene riportata la sintesi:



L'utility è stata creata su misura per BluVision, tuttavia è stata sviluppata come si trattasse di qualcosa di esterno presentandosi con un layout diverso da quello del gestionale; si è scelto infatti di non utilizzare le classi personalizzate di BluVision, ma di utilizzare quelle base di FoxPro. Per disegnare la form e creare la tabella si sono utilizzati gli strumenti di FoxPro, ricorrendo all'aiuto e alla praticità dei Designer di cui si era già avuto un po' di esperienza precedentemente. Sarebbe stato possibile crearle separatamente al progetto e poi aggiungerle, ma si è preferito procedere,

come per i programmi, dall'interno del progetto creandole all'occorrenza e secondo le esigenze del momento.

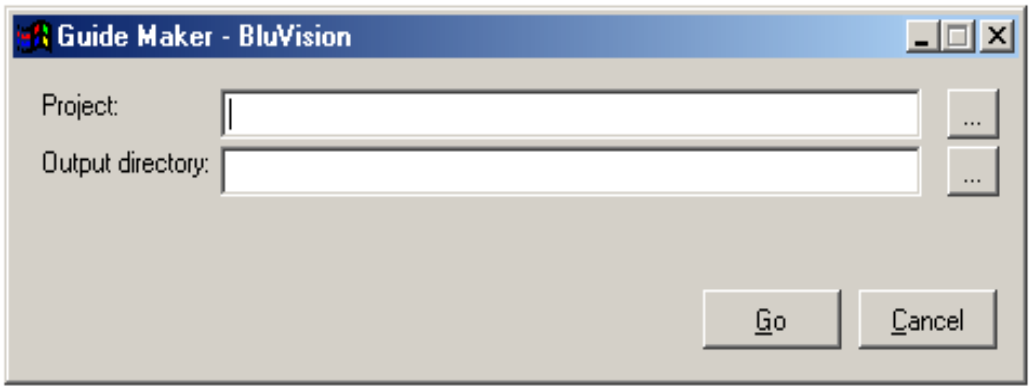
Lanciando l'utility vengono eseguite per prime le istruzioni contenute nel programma principale (evidenziato in grassetto) *go*, che contiene poche righe di codice relative alla libreria da aprire e alla form da eseguire, oltre alla definizione di alcune variabili.

```
close all
clear all
set library to foxtools
set procedure to main

public cOutDir, cPath, cNwFile

DO FORM main
```

Le variabili definite sono di tipo stringa e dovranno contenere rispettivamente il path della cartella di output, il percorso dove è salvato il progetto di BluVision e il nome dei files creati o richiamati in esecuzione. Viene dunque subito richiamata la form *main*, l'unica interfaccia tra utente e programma, che si presenta come riportato qui sotto.



Nel definirla, non sono state molte le proprietà cambiate, lasciando impostate quelle di default e limitandosi a modificarne le dimensioni e ad

aggiungere i campi necessari. Sono state aggiunte due *TextBox* dove inserire il path del progetto e quello della directory di output, stringhe che poi verranno salvate nelle variabili di cui abbiamo parlato prima, e quattro pulsanti appartenenti alla classe *CommandButton*. I primi due pulsanti, quelli affiancati alle TextBox, sono stati aggiunti per permettere la ricerca del file e della directory in una finestra di dialogo; per fare questo è stato controllato il *Click* dei due pulsanti inserendo rispettivamente

```
this.parent.projpath.value = getfile("pjx")
this.parent.outdir.value = getdir(this.parent.outdir.value)
```

e permettendo nel primo caso di ricercare i soli files con estensione pjx, mentre nel secondo di aprire una finestra di dialogo dove sono mostrate e dove è possibile selezionare la directory di output.

Per quanto riguarda gli altri due pulsanti uno rilascia semplicemente la form, mentre l'*OK* è stato gestito inserendo, anche in questo caso, nel *Click* le operazioni da svolgere.

```
cProjPath = thisform.projpath.value
cOutDir = addbs(alltrim(thisform.outdir.value))

cPath = ALLTRIM(ADDBS(JUSTPATH(cProjPath)))
DO main WITH cProjPath

ThisForm.Release
```

Le due stringhe vengono assegnate alle variabili dichiarate inizialmente, salvando però in *cProjPath* solo la directory in cui si trova il progetto; è stato fatto questo perché il tipo di informazione si è resa necessaria per aprire i singoli programmi di BluVision. Viene dunque ordinata l'esecuzione del programma *main* al quale viene passata la variabile contenente il path del progetto. Il programma main è quello che effettivamente contiene le istruzioni per entrare nel progetto e ricavare le informazioni che interessano, per le numerose righe di codice che lo compongono, è preferibile riportare di seguito solamente le istruzioni più

importanti tralasciando le procedure e le funzioni create per svolgere singole operazioni ripetitive.

```
LPARAMETERS cProjName

SELECT 0 && Viene selezionata la WorkArea più bassa
USE (cProjName) ALIAS progetto
IF not USED()
  =MESSAGEBOX("File non trovato!",16,"PDM - Errore")
  RETURN
ENDIF

SELECT * FROM progetto INTO TABLE pjx
USE IN progetto
ALTER TABLE pjx ADD COLUMN HtmlName M

SELECT pjx
SCAN ALL
  REPLACE HtmlName WITH ALLTRIM(STR(RECNO()))+".html"
ENDSCAN

USE istruzioni IN 0 ORDER NAME

DO outPage WITH "Index","Index.html"
DO outPage WITH "menu", "menu.html"
DO makePage WITH "prgMenu.html"

USE IN istruzioni
USE IN pjx
ERASE pjx.dbf
ERASE pjx.fpt
ERASE pjx.bak
ERASE pjx.tbk

MESSAGEBOX("Aprire il file 'Index.html' nella cartella di
Output per visualizzare il risultato.",0+64)

RETURN
```

Il progetto, come già visto per forms e reports, viene aperto come se si trattasse di una tabella e manipolato con le istruzioni SQL classiche. Una volta aperto, tutte le informazioni vengono salvate in una tabella *pjx* alla quale viene aggiunta subito una nuova colonna *HtmlName* di tipo *memo*,

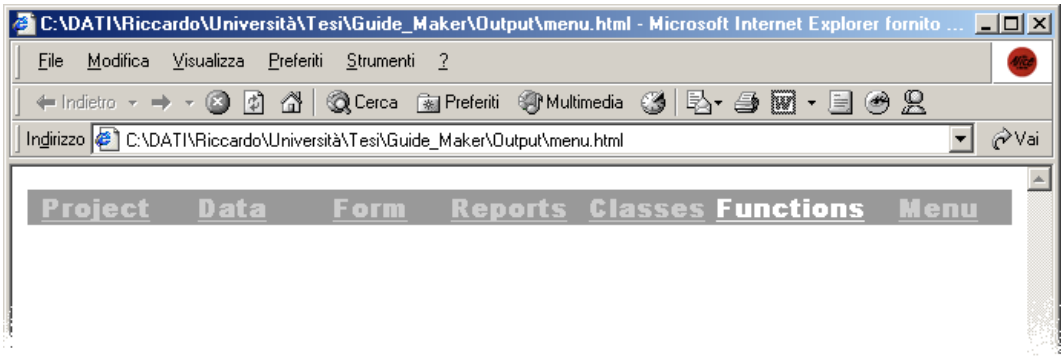
nella quale vengono inseriti dei numeri progressivi sfruttando la funzione *recno()* e ai quali viene aggiunta la stringa "*.html*" dopo che sono stati trasformati in carattere; la colonna contiene in pratica il nome che avranno le pagine di output finali. Successivamente vengono create le varie pagine, utilizzando *outPage* per la pagina principale e per il menù, mentre per le pagine contenenti le funzioni e le procedure da analizzare viene richiamata *makePage*. L'aspetto infatti che si intende dare alla pagina html di output è di un menù laterale le cui scelte corrispondono ai vari file *.prg* del progetto; a loro volta queste aprono le pagine che contengono le informazioni relative alle funzioni che si volevano. In breve, non è stata fatta un'unica pagina html con la lunga lista delle funzioni ma è stata mantenuta la suddivisione per singoli programmi; per esempio, la funzione *ChkCodDoc* creata all'interno di *genprg.prg*, nell'output ottenuto sarà contenuta nella pagina aperta dalla voce del menù corrispondente al nome del programma in cui si trova effettivamente. La pagina html dunque risulta strutturata in due frame con l'aggiunta di un altro frame in testa alla pagina come vedremo ora. Prima di passare al dettaglio della creazione delle pagine però esaminiamo le ultime istruzioni che vengono eseguite una volta create le pagine: vengono chiuse le tabelle usate e cancellata la tabella pjx creata in apertura; poi viene mandato un messaggio all'utente con indicato il file da aprire e ordinata la chiusura della form.

Come annunciato brevemente, due sono le funzioni che gestiscono la creazione delle pagine. La prima ad essere richiamata è outPage che crea la pagina principale, quella che deve essere aperta dall'utente per visualizzare l'output, e il menù in testa alla pagina.

```
PROCEDURE outPage
  LPARAMETERS cNameOP, cOutNameOP
    SELECT istruzioni
```

```
    cOutNameOP = cOutDir + cOutNameOP
  SEEK cNameOP
  IF FOUND()
    COPY MEMO memo TO (cOutNameOP)
  ENDIF
ENDPROC
```

I valori passati sono relativi al nome del campo da cercare nella tabella *istruzioni* e al nome del file che verrà creato. Questa è l'unica procedura in cui viene richiamata la tabella, che è composta di soli due records e di due soli campi, uno di tipo carattere e l'altro di tipo memo, nei quali sono stati inseriti rispettivamente il nome che deve essere cercato dalla procedura e le istruzioni html che devono essere inserite all'interno della pagina. La procedura outPage ha il compito di copiare il contenuto del campo memo all'interno del file e di crearlo se questo non esiste. Il tutto dunque si risolve nel comando COPY MEMO che crea un file il cui nome è contenuto nella variabile cOutNameOP e in esso vi copia il contenuto del field memo corrispondente. Una volta eseguita la procedura, nella cartella di output avremo i due files *Index.html* e *menu.html* creati in base alle istruzioni copiate dal corrispondente campo della tabella. Il file Index contiene semplicemente le istruzioni relative alla suddivisione in frame della pagina e all'assegnazione dei rispettivi nomi, utilizzati successivamente per richiamare le altre pagine html. Nel primo frame in alto è inserita sempre la pagina menu.html, mentre gli altri due frame *sottosx* e *sottodx* sono controllati dalle scelte di menù. La pagina menu.html invece, è una semplice tabella di una riga e sette colonne con l'aspetto che viene mostrato nella pagina seguente.



Il programma è stato creato per ricavare le sole informazioni relative ai programmi, di conseguenza l'unica voce della quale è disponibile un output è quella relativa alle funzioni ed è evidenziata col colore bianco, per le altre il colore è un grigio chiaro. La tabella creata contiene nelle singole celle dei collegamenti ad altre pagine, che tuttavia ad eccezione della sola cella "Function" sono collegamenti a pagine che non esistono. Cliccando dunque sull'unico collegamento funzionante, si richiama la pagina *prgMenu.html* che in base alle istruzioni html deve essere aperta nel frame "sottosx". Come già detto quest'ultima pagina è una sorta di menù le cui voci corrispondono ai nomi dei file prg di BluVision, vediamo dunque come è stata creata.

La procedura richiamata è makePage e la variabile che viene passata è il nome che deve avere la pagina, cioè *prgMenu.html*. Per creare e chiudere il file, sono state scritte due piccole funzioni che rispettivamente si chiamano CreaFile e ChiudiFile e al loro interno è stata inserita la sola istruzione di FoxPro fcreate e fclose. La stessa cosa è stata fatta per inserire le istruzioni html nel file creando la funzione AddText che riceve in input una certa stringa e utilizza il comando fwrite per copiarla. Anche se di piccole dimensioni la scelta di scrivere le tre funzioni è parsa molto buona perché sono numerose le volte che queste vengono richiamate, soprattutto per Addtext che nella singola pagina html viene eseguito ogni volta che è

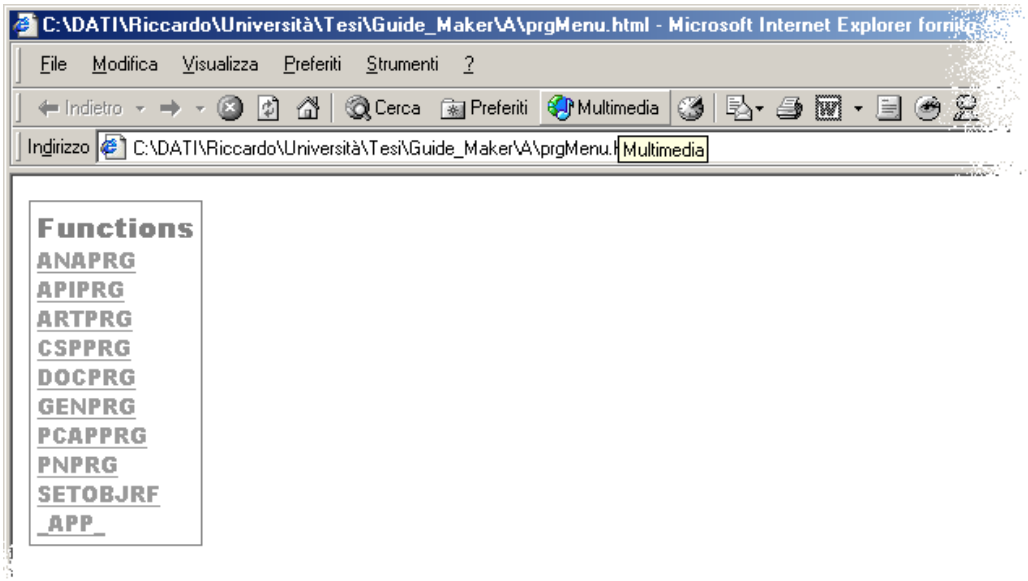
aggiunta una riga di codice. A questo punto possiamo vedere quali operazioni svolge la procedura makePage:

```
PROCEDURE makePage
  LPARAMETERS cNameMP && prgMenu.html
  LOCAL ARRAY aProjs(1)
  LOCAL nRecNo, i, cLine
  DO CreaFile WITH cNameMP
  AddText('<body link="#999999" vlink="#999999"
  alink="#999999">')
  AddText('<div align="left"><table border="1"
   bordercolor="#808080"><tr><td bordercolor="#FFFFFF">')
  AddText('<font face="Arial Black" size="3"
   color="#808080"><b>Functions</b></font><br>')
  SELECT key, name, HtmlName FROM pjx WHERE type == "P"
   INTO ARRAY aProjs
  nRecNo = _TALLY
  IF nRecNo <> 0
    =ASORT(aProjs,1)
    FOR i = 1 TO nRecNo
      cLine = aProjs(i,1)
      =AddText('<base target="sottodx"><a href=
       "'+aProjs(i,3)+'"><font face="Arial Black"
       size="2">'+cLine+'</font></a><br>')
    ENDFOR
    AddText('</td></tr></table></div>')
  ENDIF
  AddText('</body>')
  DO ChiudiFile
  FOR i = 1 TO nRecNo
    WAIT WINDOW NOWAIT aProjs(i,2)
    DO CreaFile WITH aProjs(i,3)
    DO prgFnct WITH FULLPATH(aProjs(i,2),cPath),
     aProjs(i,1)
    DO ChiudiFile
  ENDFOR
ENDPROC
```

Nelle prime righe della procedura viene subito richiamata la funzione CreaFile e successivamente Addtext con le quali si crea il file prgMenu.html e all'interno del quale vengono inserite le istruzioni html per definire i colori delle parole usate per i collegamenti e quelle per definire le caratteristiche della tabella in cui inserirli; la tabella avrà un'unica cella e i

vari collegamenti sono inseriti uno sotto all'altro, nella prima riga però è inserito in grassetto il "titolo" del menù (*Functions*).

Si procede poi selezionando dalla tabella *pjx*, che era stata creata all'inizio, solo tre campi dei records con il field *type* uguale a "P", cioè quei records contenenti le informazioni dei soli programmi, e i campi selezionati (*key*, *name* e *HtmlName*) vengono inseriti all'interno dell'array *aProjs.* Se il numero di records processati è diverso da zero, si procede ordinando l'array secondo il primo campo e, con un ciclo per ogni record, alla creazione del collegamento. La caratteristica comune del codice html che è aggiunto ad ogni ciclo, è che deve aprire il collegamento nel frame "sottosx", mentre a variare sono ovviamente l'argomento di *href* e *cLine.* La singola riga di codice risulta dunque formata dalla key, che rappresenta il nome che comparirà nella pagina html e che coincide con il nome del programma a cui fa riferimento, e dalla stringa salvata all'inizio nel field *HtmlName* che corrisponde al nome delle pagine html contenenti le procedure e le funzioni di BluVision. Alla fine del ciclo si richiamano la funzione AddText, per aggiungere le istruzioni di chiusura della tabella e della pagina, e la procedura ChiudiFile per chiudere *prgMenu.html*. L'aspetto che avrà la pagina sarà quello mostrato qui di seguito.



A questo punto non rimane che creare le pagine che effettivamente riportano le informazioni desiderate, cioè tutte quelle funzioni e procedure all'interno dei *.prg* che sono state create dai programmatori e non appartengono a FoxPro. Anche in questo caso, con un ciclo viene ripercorso l'array e creato di volta in volta un file html con nome uguale alla stringa contenuta in HtmlName e che verrà salvato nella cartella di output. Ad ogni ciclo, dopo aver creato il file viene richiamato il programma *prgFnct* al quale sono passate le variabili che contengono il path del *prg* da analizzare e la key. Vediamo ora quali sono le istruzioni che sono eseguite in ogni programma aperto:

```
LPARAMETERS cLocation, cPrgName
LOCAL cLine, cMark, cLine2, nFH, i, n;

IF not FILE(cLocation)
  RETURN
ENDIF
nFH = FOPEN(cLocation)
i=0
n=0

=addtext("<font face='Courier New' size='2'>")
=addtext("<H3><center>"+cPrgName+"</center></H3>")
```

```
DO WHILE not FEOF(nFH)
  cLine = FGETS(nFH)
* Se inizia con PROC o FUNZ
  IF UPPER(SUBSTR(cLine,1,4))=="PROC" OR
   UPPER(SUBSTR(cLine,1,4))=="FUNC"
      i=1
      cMark = cLine
      IF AT("&",cMark)#0
      cMark=SUBSTR(cMark,1,AT("&",cMark)-1)
*Tolgo dal nome della funz il commento che inizia con &&
      ENDIF
  ENDIF
* Se inizia con ENDP o ENDFU
  IF UPPER(SUBSTR(cLine,1,4))=="ENDP" or
   UPPER(SUBSTR(cLine,1,5))=="ENDFU"
      addtext("</P></LI>")
  ENDIF
* Se inizia con **!**
  IF SUBSTR(cLine,1,5)="**!**"
    IF i=1
      addtext('<LI><B>'+cMark+'</B><P>')
      i=0
    ENDIF
      n=1
      cLine2 = CheckString(cLine," "," ")
      cLine2 = CheckString(cLine2,"**!**","")
      * tolgo '**!**' dalla riga
      cLine2 = CheckString(cLine2,"*","")
      * tolgo la fila di caratteri '***' a fine stringa
      cLine2 =
       CheckString(cLine2,CHR(9),REPLICATE(" ",4))
      =addText(cLine2+"<BR>")
  ENDIF
ENDDO

IF n#1
  addtext("<center><font color='#FF6666'>Nessuna funzione
   o procedura trovate!</font></center>")
ENDIF
addtext("<hr width='70%' color='#999999'>")
addtext("</font>")
FCLOSE(nFH)
```

Il programma apre il file, e il valore ritornato relativo all' handle number del file viene salvato nella variabile *nFH*, questo viene fatto perché successivamente il valore sarà richiesto da una funzione di FoxPro. Anche

qui inizialmente sono aggiunte con *AddText* le istruzioni html relative al *font* e al titolo che comparirà in testa alla pagina html, successivamente si procede però ad analizzare le singole righe contenute nel file prg alla ricerca di quelle di interesse. In particolare ricordiamo che secondo le regole interne di NikeZadit, una funzione creata dai programmatori all'interno di BluVision doveva essere commentata opportunamente iniziando con i caratteri **!** e in testa alla funzione. Ad esempio creando la funzione *CodIvaDoc* come segue:

```
FUNCTION CodIvaDoc(cCodCliFor, cCodArt)
**!** CodIvaDoc: Ritorna codice iva della riga *********
**!** Parametri in ingresso:  - cCodCliFor: Codice Cli/For
**!**                 - cCodArt: Codice articolo
**!** Parametri in uscita:      - cCodIva: Codice Iva
```

Sfruttando questo modo di scrivere le funzioni, se pur con un procedimento un po' contorto si è riusciti alla fine a ricavare le informazioni che si desideravano, ottenendo per ogni funzione l'output mostrato qui sotto e relativo alla funzione portata in esempio:

- **FUNCTION CodIvaDoc(cCodCliFor, cCodArt)**

```
CodIvaDoc: Ritorna codice iva della riga
Parametri in ingresso:- cCodCliFor: Codice Cli/For
                     - cCodArt: Codice articolo
Parametri in uscita:- cCodIva: Codice Iva
```

Il programma esegue un *fget* per ogni riga del programma salvando i caratteri in una variabile cLine, fino a che non arriva alla fine del file. Quello che fa il programma a questo punto è di cercare quelle funzioni che si distinguono perché nella seconda riga iniziano con i caratteri "**!**", dunque ad ogni ciclo, utilizzando alcuni *if* e le funzioni *sustr, upper* e *at* per la manipolazione di stringhe, la variabile cLine è sottoposta ad alcuni controlli per vedere se rientra tra le stringhe di interesse. La scelta adottata è di presentare le funzioni come un elenco, aggiungendo dunque al codice

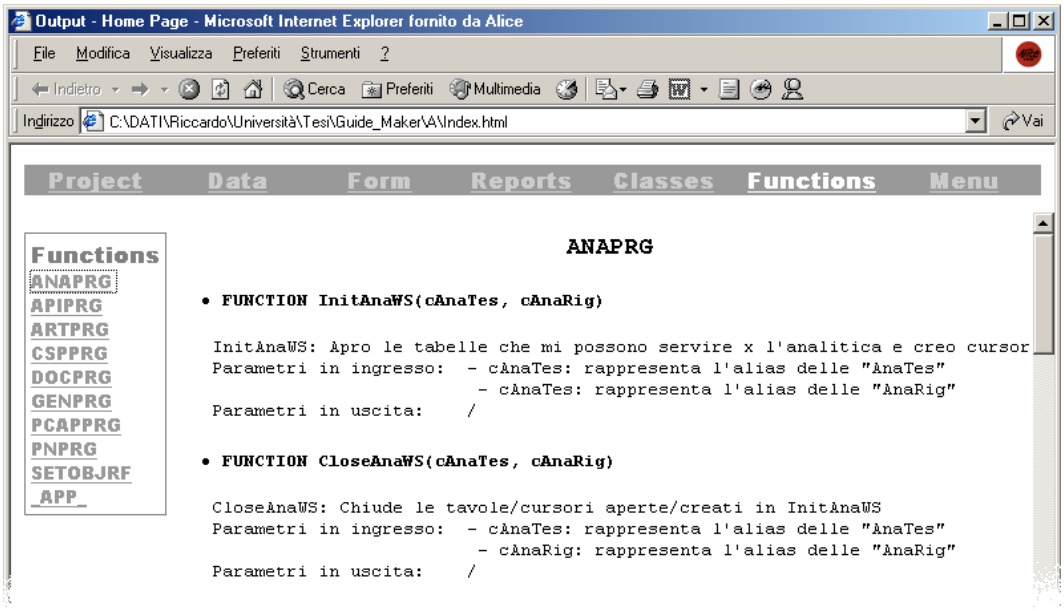
html l'istruzione <*li*> solo nella prima riga tra quelle appartenenti ad una singola funzione.

Per selezionare le sole funzioni, si è reso necessario salvare la riga che iniziasse con "proc" o "func" in una variabile *cMark*, e di stamparla solo se quella successiva iniziava con **!**; è stata poi aggiunta una variabile di controllo *i* per stampare una sola volta la stringa in cMark, visto che le righe commentate per singola funzione erano di norma almeno tre. Se la condizione era soddisfatta dunque si ottenevano in output le righe desiderate, opportunamente modificate, e con l'aspetto mostrato nell'esempio sopra. Alcune righe del "tipo" cMark, avevano dei normali commenti alla fine che non si volevano riportare e sono stati tolti, mentre per le cLine candidate ad essere riportate nella pagina di oitput è stata creata la funzione CheckString che sostituisce all'interno della stringa alcuni caratteri con altri: è stata utilizzata in particolare per togliere i caratteri di commento che le distinguevano, per eliminare dei caratteri "*" presenti alla fine di alcune cLine e per migliorare il più possibile il layout con cui si sarebbe poi presentata nella pagina di output.

Ogni volta invece che la stringa in cLine iniziasse con "*endfu*" o "*endp*" veniva "chiuso" il punto di elenco inserendo nella pagina le istruzioni ; si è notato però che questa operazione veniva fatta anche per quelle stringhe che effettivamente non interessavano, cioè l'istruzione veniva aggiunta anche per gli *endfu* o *endp* delle funzioni che non si volevano riportare. Il problema poteva essere risolto aggiungendo un'altra variabile di controllo come quella per cMark, che eseguisse le istruzioni solo se appartenente alle funzioni di interesse.

A fine programma, le ultime istruzioni inserite riguardavano un messaggio nel caso non fossero state trovate funzioni, e in ogni caso l'aggiunta di una linea e di un tag di chiusura *font.*

A questo punto tutte le operazioni sono state eseguite e la form viene rilasciata secondo l'istruzione nel click del pulsante *Go* dopo aver avvisato l'utente che il risultato è visionabile aprendo il file Index.html nella cartella di output. Un esempio di come appare la pagina html finale è riportato qui sotto:



8. Considerazioni Finali

Un giudizio finale sullo stage svolto può essere dato considerando quali fossero le aspettative iniziali dello studente e confrontandole con l'effettivo contributo che l'esperienza ha dato. Lo stage prevedeva l'utilizzo di un software e di un linguaggio di programmazione nuovi e dei quali non si avevano esperienze precedenti, tuttavia le basi e la somiglianza con uno dei linguaggi trattati durante il Corso di Laurea hanno permesso un buon inserimento nel nuovo ambiente di lavoro. L'impressione che si è avuta di Visual FoxPro è positiva, anche se non è possibile dare un giudizio obiettivo confrontandolo con programmi simili o dello stesso importanza; si è rilevato tuttavia un software completo con strumenti semplici e adatti alle esigenze delle attività svolte. In particolare il Form Designer, usato in più occasioni e che ha facilitato il lavoro di creazione ed inserimento di oggetti all'interno della form, e Trace, utilizzato parallelamente al Fox Editor, che si è rilevato molto utile per localizzare e risolvere velocemente eventuali errori di programmazione. Il giudizio dato fa riferimento all'utilizzo del software limitatamente allo svolgimento delle attività assegnate, che sono da considerarsi di modesta importanza se rapportate alle capacità dei programmatori dell'azienda, di conseguenza un giudizio più preciso può essere dato al sorgere di problematiche e di limiti di Visual FoxPro nel curare un progetto di dimensioni piuttosto grandi come il gestionale BluVision. Sugli altri strumenti non appartenenti a Visual FoxPro ritorniamo per esprimere un giudizio positivo sia per la semplicità di utilizzo, sia perché rappresentano un buon metodo di lavoro dal punto di vista organizzativo; scelta tuttavia che mal si concilia col fatto di lavorare su di un software che offre certificazione solo legata alle personalizzazioni.

Argomento questo già trattato precedentemente ma che non trova una spiegazione logica.

Per quanto riguarda invece gli obiettivi da raggiungere secondo la pianificazione iniziale, possiamo dire che nonostante questa non sia stata rispettata, le attività assegnate sono state tutte completate e, per la quasi totalità del tempo, in modo autonomo e in maniera soddisfacente, segnale dunque di una certa padronanza ottenuta a fine del periodo di stage per quanto riguarda gli strumenti utilizzati. Tuttavia pur essendo un software dal facile utilizzo, Visual FoxPro risulta uno strumento complesso se si pensa alle svariate funzioni e utilizzi che ha, dunque solo dopo un adeguato periodo di apprendimento si può acquisire una padronanza al pari dei programmatori della azienda ospitante.

Nonostante però i risultati siano da considerarsi buoni, è sembrato inutilmente lungo il periodo dedicato allo studio di manuali e soprattutto il tempo dedicato alla comprensione della struttura alla base del progetto BluVision. È risultata praticamente assente la supervisione del Tutor Aziendale o di qualcuno che presentasse sia il software da utilizzare che quello al quale prestare manutenzione, di conseguenza il periodo precedente allo svolgimento delle attività poteva essere ridotto al minimo affiancando allo studente una persona che insegnasse le basi degli strumenti da utilizzare.

Lo stesso si può dire per il periodo in cui sono state svolte le attività assegnate, nel quale non veniva data sufficiente attenzione nel presentare il lavoro da svolgere; è il caso della form *I_tsk* che è stata scritta e corretta in più fasi prima di ottenere un risultato concreto perché non erano ben chiare quali fossero le situazioni da gestire e da tenere in considerazione. D'altra parte, questo è servito per comprendere meglio la struttura di BluVision e

per integrare man mano le conoscenze sul linguaggio di programmazione, e inoltre, per essere spronato in certe situazioni a risolvere e a prendere decisioni autonomamente, come nel caso della gestione delle password, dove le modifiche al codice richiedevano un attenta analisi dei fattori che incidevano sul nuovo modo di criptare le password e dei cambiamenti che avrebbe introdotto la modifica. Infine, una certa soddisfazione si è avuta con l'ultima attività svolta, curando un progetto dall'inizio alla fine e vedendolo effettivamente funzionare e produrre i risultati desiderati.

Nel complesso dunque si tratta di un'esperienza utile che è servita a valutare il grado di adattamento ad un ambiente di programmazione nuovo e ad un linguaggio sconosciuto acquisendo una certa elasticità sicuramente necessaria in futuro.

9. Appendice

- **Manuale FoxPro**

  Si tratta di una guida in formato digitale contenente la traduzione dall'edizione inglese di "User Guide Visual FoxPro®" di Microsoft.

- **Raccolta Esempi**

  Documento in formato digitale con una raccolta di esempi estrapolati dall'ultima versione in italiano della guida in linea di Visual FoxPro, che pur rifacendosi alla versione 3.0 è un utile strumento per imparare l'uso di comandi e funzioni del linguaggio.

- **http://www.foxitaly.com**

  Sito in lingua italiana dedicato ai programmatori in Microsoft FoxPro e Visual FoxPro®

- **http://www.NikeZadit.com**

  Sito della Ditta ospitante con le informazioni dell'azienda, sia di tipo organizzativo che riguardanti l'organico, e nel quale vengono presentati i prodotti e i servizi offerti.