UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA
DELL'INFORMAZIONE
CORSO DI LAUREA IN INGEGNERIA INFORMATICA

DEBUG DI UN SOFTWARE PER IL CONTROLLO E LA GESTIONE DI CENTRALI ANTINTRUSIONE

Laureando
Alberto Mazzucco

Relatore
Marcello Dalpasso

Anno Accademico 2011/2012

SOMMARIO

Questo documento si propone di illustrare il lavoro svolto durante lo stage di cinquecento ore presso l'azienda Fracarro Radioindustrie S.p.A. di Castelfranco Veneto e attuato a partire dalla data del 10 Marzo 2011 fino al 5 Agosto 2011.
Il Gruppo Fracarro è un insieme di aziende di riferimento internazionale nel settore delle telecomunicazioni e in quello della sicurezza. Tra queste, la Fracarro Radioindustrie S.p.A., è il perno attorno al quale ruota tutto il gruppo e possiede un settore di ricerca e sviluppo impegnato continuamente in nuovi progetti.

Il progetto che mi è stato proposto e in cui successivamente sono stato inserito si trovava, al mio arrivo, in una fase intermedia e prevedeva l'aggiornamento del software Programmer Sicurezza creato appositamente per la configurazione ed il controllo da pc di un insieme di dispositivi anti intrusione brevettati dall'azienda.
L' applicativo in questione, già commercializzato, era stato realizzato principalmente in Visual Basic 6 e il rinnovamento pianificato consisteva nel passaggio ad una versione più recedente del software di sviluppo, ovvero Visual Basic.NET, fornito nella suite Microsoft Visual Studio. Al momento del mio inserimento nel progetto la fase di trasferimento del codice sorgente era quasi completamente ultimata ed era stata affidata a personale esterno all'azienda il quale si era limitato alla sola conversione del software.

L'obbiettivo dell'attività di tirocinio svolta è stato quindi quello di portare avanti parallelamente la ristrutturazione del programma e in particolare mi è stato proposto di testare la parte di interfaccia per le diverse tipologie d'utenza, di trovarne le debolezze e gli errori che non ne consentivano un corretto funzionamento e di correggere quest'ultimi andando prima a studiare e conseguentemente a modificare le parti di codice da cui venivano generati. Dopo aver reso stabile e senza interruzioni inattese la parte del software appena descritta, è stato necessario sistemare la vista grafica del programma che, a causa della conversione, era stata parzialmente alterata o compromessa.
Una volta completata questa prima fase di ottimizzazione del software, lo stage è proseguito con i test relativi alla connessione e allo scambio di messaggi tra un dispositivo ed il programma stesso e con la successiva correzione degli errori critici scoperti durante queste nuove prove.

In questa relazione verranno descritti innanzitutto i dispositivi e il software interessati nel progetto, le varie fasi ed operazioni affrontate nel corso dell'attività di stage e gli strumenti, i linguaggi di programmazione e le tecnologie usate.

INDICE

ELENCO DELLE FIGURE

1 | INTRODUZIONE

1.1 L'AZIENDA

La Fracarro Radioindustrie S.p.A. di Castelfranco Veneto è un'azienda fondata nel 1933 e divenuta negli anni una delle più importanti realtà europee nel campo della ricezione e distribuzione dei segnali audio video e della sicurezza attiva.
L'unione di questi due settori di attività ha consentito a Fracarro di proporre soluzioni integrate per la realizzazione di edifici e abitazioni sicure e tecnologiche da dove è possibile collegarsi ed interagire attivamente con il mondo.
Nel settore audio video l'azienda offre un catalogo completo di prodotti in grado di soddisfare le più svariate problematiche presenti nel campo delle telecomunicazioni: antenne terrestri e satellitari, centrali di testa, amplificatori, miscelatori, alimentatori, centralini e componenti per la distribuzione.
L'azienda inoltre progetta e produce sistemi antintrusione cablati e senza fili, sistemi per videosorveglianza e soluzioni per la protezione degli impianti fotovoltaici.
La progettazione di tali apparati avviene nel moderno laboratorio di ricerca e sviluppo di cui è dotata, il quale ha permesso la creazione di soluzioni e di dispositivi altamente affidabili in grado di risolvere le più svariate esigenze degli utenti. Particolare attenzione viene posta ai prodotti già commercializzati sia fornendo un servizio continuo di assistenza a clienti ed installatori, sia correggendo eventuali problemi di progettazione o produzione.
Infine un ulteriore accorgimento adottato dall'azienda è quello di rimanere sempre attenta alle nuove problematiche create dall'evoluzione del mercato o dalla nascita ed esplosione di nuove tecnologie in modo da poter modificare o adeguare i propri prodotti per riuscire in questo modo a rimanere competitiva e aggiornata.

Figura 1: Logo Aziendale

1.2 I PRODOTTI SICUREZZA

Nel settore della sicurezza la Fracarro ha sviluppato negli anni una serie di prodotti sempre più evoluti in grado di soddisfare le crescenti esigenze della clientela e aumentando le possibili combinazioni e soluzioni d'installazione.
I prodotti di questo specifico ramo dell'impresa si suddividono principalmente in due sottocategorie: quella delle soluzioni antintrusione e quella delle soluzioni per videosorveglianza.
Per quanto riguarda le prime, fanno parte di questa categoria tutti quei dispositivi che compongono un comunissimo impianto di sicurezza. Troveremo di conseguenza: centraline, rivelatori, sirene, combinatori telefonici, chiavi elettroniche, dispositivi di comando e molti altri accessori che aumentano le potenzialità dell'impianto stesso.
Analizzando la seconda invece possiamo trovare: telecamere, videoregistratori, tastiere di controllo remoto, hard disk per la memorizzazione dei filmati, quad, monitor, illuminatori e tanti altri complementi per accrescere il livello di sicurezza dell'impianto oppure ottimizzarne determinate caratteristiche.
In questa relazione verranno presi in considerazione solamente i dispositivi che fanno parte della prima delle due categorie appena citate, e in maggior dettaglio, le centraline antintrusione, le quali sono state l'oggetto di studio su cui si è basato il periodo di formazione.
Le centraline con il quale si è dovuto lavorare durante il tirocinio corrispondono ai seguenti modelli: CE64T, CE16T, Solution16 e Defender. Quest'ultima in particolare è disponibile in diverse versioni e più precisamente Defender 8, Defender 12, Defender 64 e Defender 64T.

Figura 2: Centrale Defender con sirena da esterno e rivelatore a doppia tecnologia

Le diversità tra le varie tipologie di dispositivo interessano soprattutto i componenti e la tecnologia utilizzata, questo dovuto al fatto che i diversi progetti sono stati elaborati a distanza di alcuni anni. Il salto temporale tra i primi tre modelli e quelli della categoria Defender si aggira infatti attorno ai cinque anni. Altre differenze sostanziali si misurano in termini di portata ovvero del numero massimo di ingressi potenziali presenti sulla centralina e fruibili al momento dell'installazione.

1.3 IL SOFTWARE PROGRAMMER SICUREZZA

Figura 3: Supporto cd-rom con il quale viene fornito il software Programmer Sicurezza

Il software di controllo e gestione della famiglia di centrali antintrusione analizzate nel capitolo precedente e progettato dall'azienda a seguito della produzione di questi dispositivi è denominato Programmer Sicurezza.
Questo programma viene dato a corredo su supporto CD-ROM (figura 3) con l'acquisto di una centrale, è installabile ed eseguibile da pc e, tramite l'utilizzo di un apposito modulo usb (figura 4), rende possibile la comunicazione locale tra il dispositivo e la piattaforma sul quale viene installato il Programmer Sicurezza. Il software inoltre si autoconfigura in base al modello di centrale ed è gestibile anche in modalità remota, utilizzando un modem telefonico standard collegato alla linea PSTN. E' stato realizzato per diversi scopi, tra i quali il monitoraggio e la creazione di impianti virtuali, ma anche per permettere all'utente la lettura della configurazione memorizzata all'interno di una centrale e la programmazione di tutte le sue funzioni.

Analizzando più accuratamente la storia della nascita e creazione di questo software essa risulterà molto particolare. L'azienda infatti aveva inizialmente commissionato la sua produzione ad una ditta esterna, la quale una volta arrivata ad una buona percentuale di avanzamento del progetto ha consegnato quanto prodotto fino a quel momento e non lo ha più completato per la scarsità di informazioni che aveva sul settore applicativo nel quale il software doveva operare. Il programma allora è stato ripreso in mano e completato da personale interno alla Fracarro, ha attraversato una fase obbligatoria di test per poi essere infine commercializzato.

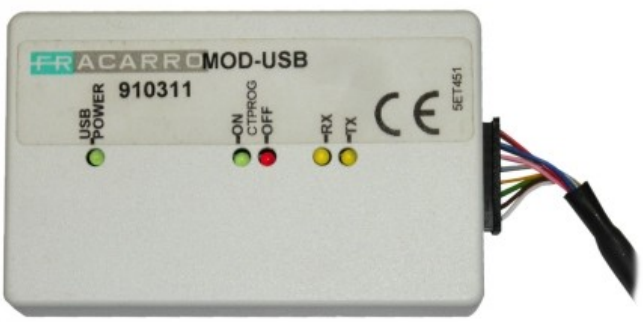

Figura 4: Modulo USB per il collegamento delle centraline

Il software era stato realizzato quasi interamente in Visual Basic 6 e per una percentuale ben più ridotta in C. Questa seconda parte realizzava il meccanismo di interfaccia tra programma e dispositivo, gestiva lo scambio e la lettura dei messaggi ed impostava le tempistiche per evitare la perdita di pacchetti durante la trasmissione o ricezione.

1.4 L'EVOLUZIONE

Dopo alcuni anni di commercializzazione, in seguito allo sviluppo di nuovi progetti per potenziare le funzionalità delle centraline di ultima generazione (famiglia Defender) l'azienda ha quindi deciso di effettuare parallelamente anche un rinnovamento del software di comunicazione.
La scelta intrapresa dalla Fracarro è stata decisa per diversi motivi: in primo piano la necessità di eliminare errori sistematici presenti nel codice, rilevati e

comunicati dagli utenti nel corso degli anni di commercializzazione del software, che causavano il funzionamento non corretto del programma in certe modalità di configurazione e con determinati dispositivi; in secondo luogo il desiderio di modernizzare la versione del codice in modo da renderlo perfettamente compatibile con le nuove tecnologie informatiche presenti sul mercato; infine per poter ottimizzare alcune funzionalità del software già presenti e poter aggiungerne di nuove.

L'azienda ha quindi scelto di passare alla versione .NET del linguaggio, precisamente Visual Basic.NET, fornita nel pacchetto Microsft Visual Studio e che utilizza il .NET Framework. Il passaggio predisposto non si presentava tra i più semplici, questo perché le due versioni del software di sviluppo sono contraddistinte da notevoli diversità di funzionamento che non rendono possibile la totale compatibilità di programmi scritti nel vecchio linguaggio. Nonostante la presenza di un tool per la migrazione del software alcune funzionalità non possono essere convertite automaticamente e richiedono un trasferimento manuale alla nuova versione.

Valutando l'impegno e le conoscenze richieste il personale a capo del progetto ha di conseguenza deciso di assegnare ad un programmatore esterno il compito di realizzare le modifiche preventivate in modo da poter proseguire con lo sviluppo dei progetti in cui era già impegnata senza privarli di forza lavoro. La scelta è ricaduta su una figura altamente specializzata nel campo della programmazione e che avesse buone conoscenze nell'ambiente descritto.

2 ANALISI DEL PROGETTO

2.1 GLI OBBIETTIVI

Al momento dell'inserimento nel progetto buona parte della conversione del codice era già stata effettuata e rimaneva solamente la parte inerente al meccanismo di interfaccia realizzata in C di cui si è parlato in precedenza. L'obiettivo da raggiungere inizialmente è stato quello di eliminare gli errori presenti nel nuovo software e una volta completata questa parte di ottimizzarlo dal punto di vista grafico. Dopo aver finito questa prima fase del lavoro e dopo che la conversione del software è stata definitivamente completata dall'analista esterno, il nuovo proposito dell'esperienza è stato quello di eseguire i test di connessione tra il nuovo software in modalità debug e alcuni dispositivi antintrusione. In questo secondo stadio il lavoro doveva trattare l'eliminazione dei nuovi errori riscontrati e l'ottimizzazione dello scambio dei messaggi in modo che la connessione avvenisse nel minor tempo possibile.
Analizzando i due step principali del progetto si può capire quindi che l'obbiettivo ricercato maggiormente è stato quello di debuggare (eliminare gli errori) il software nel modo più completo possibile.

2.2 PANORAMICA

Lo sviluppo, la modifica e l'aggiornamento di un software sono attività molto complesse, e come tali inevitabilmente soggette ad errori; in particolare, si possono evidenziare due grandi classi di problemi all'interno dei processi appena citati: quelli derivanti da una specifica dei requisiti inesatta, ed i difetti nell'implementare una corretta specifica dei requisiti. I primi vengono introdotti sin dall'inizio dello sviluppo (fase di analisi) e come tali sono estremamente dannosi, in quanto si ripercuotono su tutte le successive fasi (modellazione, implementazione) sino a quando non emergono dalla mancanza del sistema a realizzare alcuni compiti. I secondi vengono introdotti in fasi più avanzate, quindi di design o codifica, in seguito a cause differenti, che vanno dalla scarsa

conoscenza del linguaggio, ad assunzioni non verificate, alla complessità del progetto, sino alla semplice distrazione.
Alcuni di questi errori talvolta non provocano conseguenze catastrofiche mentre la maggior parte delle volte non permettono al software di svolgere le azioni programmate e di conseguenza risulta necessario intraprendere l'azione di debug.

2.2.1 IL DEBUGGING

Il termine debugging sta ad indicare l'attività che consiste nell'individuazione della porzione di codice in cui è presente uno o più errori (bugs) rilevati nei software a seguito dell'utilizzo degli stessi.
L'errore può essere localizzato sia in fase di collaudo del programma, quando cioè questo è ancora in fase di sviluppo e non è ancora pronto per essere utilizzato dall'utente finale, sia in fase di utilizzo del programma da parte di quest'ultimo.
Dopo aver riscontrato l'errore segue la fase di debugging, ossia di individuazione della parte di software, a volte molto complesso, nella quale si nasconde l'errore.
Oggigiorno questa attività è supportata da applicazioni specifiche (debugger), che mostrano al programmatore l'esecuzione, istruzione dopo istruzione, del software, permettendo nello stesso tempo la visione e l'analisi degli input e output del programma stesso.
Prima che fossero disponibili tali strumenti per l'attività di individuazione e correzione degli errori e ancora adesso in mancanza di essi, si ricorre alla più semplice ma anche meno efficace tecnica di visualizzazione a video o stampa su file delle istruzioni che il programma sta eseguendo passo dopo passo, inserendo a tal scopo nel codice delle istruzioni apposite.
Il debug è una delle operazioni più importanti per la messa a punto di un programma, spesso è estremamente difficile per la complessità del software che si sta sviluppando e delicata per il rischio di introdurre nuovi errori o comportamenti non in linea con quelli desiderati nel tentativo di correggere quelli per cui si è intrapresa l'attività stessa.
Sebbene ogni volta che si ricorre al debug per perfezionare un software, tale compito sia unico e costituisca una storia a sé, alcuni principi generici sono sempre applicabili. In particolare, nell'ambito di applicazioni software, in genere si possono riconoscere cinque fasi riassunte nello schema in Figura 5.

Figura 5: Schema evolutivo del processo di debug

2.3 APPROCCIO AL PROGETTO

Il mondo delle centraline antintrusione Fracarro è un argomento molto ampio che racchiude l'ingegno e gli anni di lavoro di molte persone. Addentrarvisi non è cosa semplice e immediata e richiede inoltre una buona conoscenza generale di una serie di materie di carattere ingegneristico quali l'elettrotecnica, l'elettronica, le telecomunicazioni e l'informatica.
Inoltre per poter riuscire ad eseguire il debug di un software nel modo più professionale possibile è necessario avere delle conoscenze specifiche su come sia stato precedentemente strutturato e creato il programma e possedere la perfetta padronanza nell'uso del software di sviluppo. Tali capacità sono imprescindibili e allo stesso tempo difficili da raggiungere rapidamente per una persona che intende intraprendere un progetto come quello in esame.
Pertanto è stato indispensabile eseguire un'attenta riflessione sulla giusta modalità di approccio al progetto tramite una disamina delle conoscenze già possedute ed il confronto con il personale interno al progetto.

2.4 PIANIFICAZIONE

A seguito del briefing iniziale avvenuto con il tutore aziendale si è quindi deciso di affrontare il progetto con un avvicinamento ad esso progressivo. Per effettuare un immersione graduale nell'ambito aziendale e in quello più particolare del software si è pensato di spendere inizialmente del tempo per accrescere le conoscenze attraverso lo studio della struttura fisica dei dispositivi e delle loro funzionalità pratiche, e l'approfondimento del software Programmer Sicurezza, del suo utilizzo operativo e del nuovo linguaggio utilizzato (VB.Net).
Dopo aver raggiunto un adeguato livello di preparazione il passo successivo è stato quello dell'avvio dei test sulla nuova versione del programma per identificare il maggior numero di bugs presenti ed una volta completata questa fase si è proseguito con quella di debug.
E' doveroso a questo punto effettuare una considerazione sui tempi previsti per svolgere il lavoro richiesto. Il debug di un software non è cosa semplice e questo implica il fatto che non sia altrettanto agevole stimarne il costo in unità di ore di lavoro. Questa considerazione aggiunta al fatto che il progetto consisteva in più

step di debug ha portato alla scelta di non pianificare inizialmente tutti i diversi passaggi della ristrutturazione richiesta dall'aggiornamento programmato. Solamente in un secondo momento, una volta completata questa prima fase, è stato possibile predisporre i nuovi interventi da eseguire.
Una volta completato il primo step infatti si è potuto intraprendere il passaggio successivo, ovvero quello di sistemare l'interfaccia grafica del software. Dopo di che, completata quest'ultima e ultimata interamente la conversione del software da parte dell'analista esterno, è stato possibile organizzare l'ultima parte del progetto, quella cioè dei test sulla connessione in locale tra il nuovo software e alcuni dispositivi ed il debug dei nuovi errori riscontrati.

3 | STRUMENTI E TECNOLOGIE

3.1 VISUAL BASIC

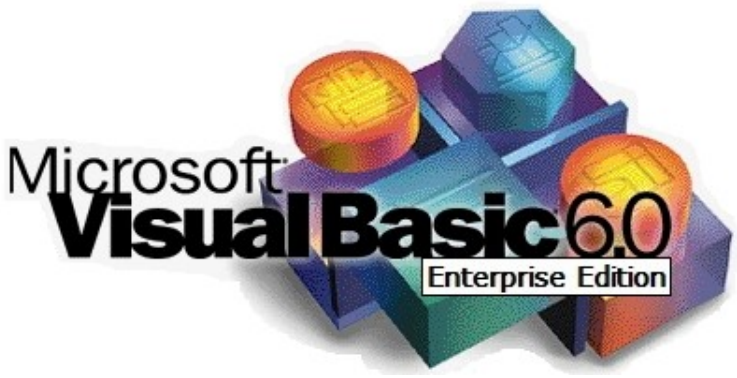

Figura 6: logo Visual Basic 6.0

Visual Basic, chiamato anche in forma abbreviata VB, è il linguaggio scelto dall'azienda per realizzare la prima versione del software Programmer Sicurezza; la versione corrente a quel tempo era la 6.0. É un linguaggio di programmazione event driven (programmazione ad eventi) arrivato fino alla versione 6, la cui sintassi deriva dal Basic. Le principali caratteristiche che contraddistinguono VB sono:

- la semplicità d'uso: non utilizza la tipica formalità di punteggiatura di quasi tutti gli altri linguaggi;
- il suo ambiente di lavoro RAD (Rapid Application Development) permette di realizzare in breve tempo interfacce grafiche per utenti (GUI – Graphic User Interface) anche complesse;
- il pratico accesso alle basi di dati;
- la creazione di controlli ActiveX con il linguaggio stesso.

Tramite l'integrazione con appositi controlli (VBX e OCX) e collegamenti (OLE) presenti nell'ambiente di lavoro o realizzati da altri

programmatori, è possibile aggiungere potenzialità al linguaggio, aggiungendo nei propri progetti nuove funzioni o ampliando funzioni già esistenti.
Alcune peculiarità di VB sono l'opportunità di personalizzare il limite superiore ed inferiore degli array e la possibilità di eseguire un'applicazione senza effettuare una compilazione completa, in questo modo risulta possibile modificare parte del codice e continuare l'esecuzione direttamente in modalità di debug.
Le prime versioni di VB non supportavano la programmazione orientata agli oggetti, ma dalla versione 4.0 questa possibilità è stata introdotta anche se in maniera limitata poiché non era possibile implementare l'ereditarietà delle classi. Altri due aspetti negativi di questo linguaggio rispetto a molti altri consistono nella gestione limitata dei puntatori e nell'assenza dei numerici senza segno.

3.1.1 LA PROGRAMMAZIONE AD EVENTI

Come precisato precedentemente Visual Basic adotta la tecnica di programmazione ad eventi, uno dei molteplici paradigmi dell'informatica. Al contrario di quella tradizionale, in cui l'esecuzione delle istruzioni segue solamente percorsi prestabiliti, che si diramano in punti definiti dal programmatore, nei software codificati utilizzando questa tecnica il flusso del programma è determinato dal verificarsi di eventi esterni.
Un sistema codificato con questo tipo di approccio invece di aspettare che un'istruzione impartisca il comando di elaborare una certa informazione, è predisposto per eseguire continuamente un ciclo di istruzioni, all'interno del quale ve ne sono alcune che verificano la disponibilità dei dati da elaborare e, nel caso siano presenti, lanciano l'esecuzione della parte di programma scritta appositamente per gestire l'evento in questione.
Gli eventi esterni possono essere molteplici, tra i più comuni vi sono ad esempio la creazione di un file in una cartella oppure la pressione di un tasto sulla tastiera o sul mouse. Tali eventi possono essere rilevati mediante polling (interrogazione) eseguito periodicamente all'interno di un set d'istruzioni oppure in risposta ad un interrupt, altre volte invece si utilizza una combinazione di entrambe queste due tecniche.
Nei software che utilizzando questo paradigma sono presenti solitamente dei brevi sotto-programmi denominati event handlers (gestori degli eventi) che vengono

eseguiti in seguito al verificarsi degli eventi esterni, e un dispatcher, che effettua materialmente la chiamata. Spesso quest'ultimo utilizza una coda apposita dove vengono memorizzati gli eventi già verificatisi ma non ancora elaborati. Molte volte i gestori degli eventi possono attivare, all'interno del loro set d'istruzioni, nuovi eventi producendo così una cascata di eventi.

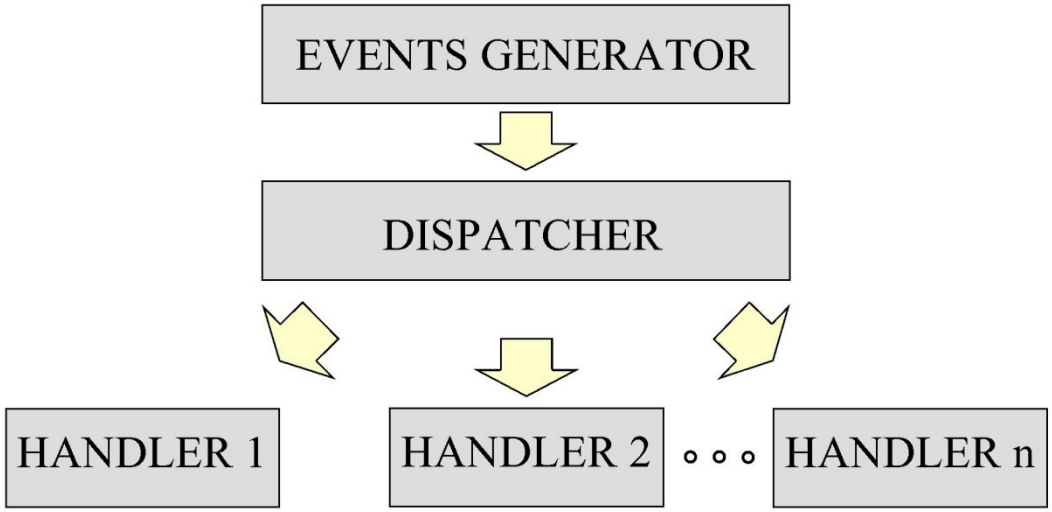

Figura 7: Schema funzionamento software event-driven

Generalmente i programmi dotati di un'interfaccia grafica sono realizzati secondo questo approccio: i sistemi operativi ne sono un esempio.
Un sistema, o programma, controllato da comandi può essere considerato un caso particolare di software event-driven, in cui il sistema, normalmente inattivo, aspetta che si verifichi un evento molto particolare, l'invio di un comando da parte dell'utente.

3.2 LA PIATTAFORMA MICROSOFT.NET

La piattaforma di sviluppo Microsoft .NET è una infrastruttura per la programmazione che funge da supporto alla suite di prodotti .NET e che permette lo sviluppo di applicazioni desktop e server. La prima versione di questo versatile strumento è stata rilasciata da Microsoft nel 2002. La sua caratteristica principale è di essere indipendente dalla versione operativa di Windows su cui è installata, e

di includere molte funzionalità progettate espressamente per integrarsi in ambiente internet e garantire il massimo grado di sicurezza e integrità dei dati.
Microsoft .NET è corredato da una serie di strumenti di sviluppo delle applicazioni, progettati in modo da funzionare in modo integrato all'interno della piattaforma. Uno dei principali strumenti è l'IDE (Integrated Development Environment cioè Ambiente di sviluppo integrato) denominato Visual Studio.
Alcune delle caratteristiche principali della piattaforma .NET sono le seguenti:

Interoperabilità: viene fornita la cooperazione e lo scambio di informazioni o servizi tra nuovi e vecchi sistemi, in maniera più o meno completa e priva di errori, con affidabilità ed ottimizzazione delle risorse. E' reso possibile l'accesso a funzionalità implementate in applicativi e meccanismi che vengono eseguiti all'esterno del ambiente .NET. È possibile inoltre accedere ai componenti COM utilizzando i namespaces System.Runtime.InteropServices e System.EnterpriseServices namespaces.

Ambiente di esecuzione comune: il CLR (Common Language Runtime) è la macchina virtuale del framework .NET. Tutti i programmi sono eseguiti sotto la sua supervisione e vengono in questo modo garantite alcune specifiche nell'ambito della gestione della memoria, della sicurezza e delle eccezioni.

Indipendenza dal linguaggio utilizzato: il framework .NET introduce un sistema di tipi comuni (CTS) , che definisce tutti i diversi tipi e costrutti supportati dal CLR e come essi possono interagire tra di loro. Grazie al CTS il .NET framework supporta lo scambio di tipi e istanze tra librerie scritte usando linguaggi .NET differenti.

Base Class Library: è una libreria, parte della Framework Class Library, accessibile da tutti i linguaggi della piattaforma .NET. Questa libreria fornisce classi che implementano funzionalità di base riguardanti la gestione dei file, la manipolazione grafica, l'interazione con i database, manipolazione dei dati XML e molto altro.

Sicurezza: .NET è stato pensato per affrontare alcune delle vulnerabilità, come il buffer overflow, che vengono sfruttare per scrivere software dannoso. Inoltre, .NET fornisce un modello comune di sicurezza per tutte le applicazioni.

Portabilità: Il Framework .NET è stato progettato per essere completamente indipendente dalla piattaforma in cui viene eseguito. Questo permette di eseguire un'applicazione scritta per girare sull'ambiente .NET in una qualsiasi piattaforma per la quale il framework è stato implementato.
Microsoft non ha implementato il suddetto framework per alcuna piattaforma diversa da Microsoft Windows ma ha reso disponibili le specifiche del CLI cosicché sia possibile per terze parti implementarlo su una qualsiasi piattaforma.

3.2.1 IL FRAMEWORK .NET

Il .NET Framework è la parte centrale della tecnologia .NET di Microsoft, ovvero l'ambiente per la creazione, la distribuzione e l'esecuzione di tutti gli applicativi che supportano .NET siano essi servizi web o altre applicazioni.
Il Framework .NET è composto da diverse parti che possono essere riassunte nello schema in Figura 8.

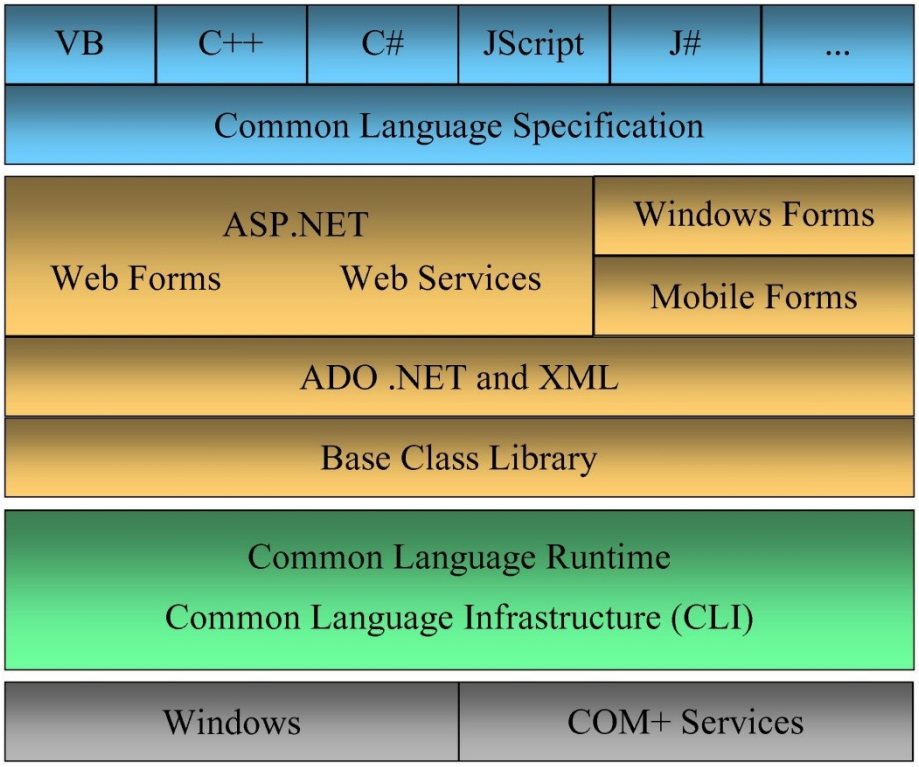

Figura 8: Architettura Framework .NET

Per quanto riguarda i compilatori, nella sua versione per sviluppatori (SDK), .NET contiene i compilatori per C#, Visual Basic .NET, JavaScript, J#. Oltre a questi linguaggi, forniti da Microsoft, sono utilizzabili altri linguaggi, come ad esempio Delphi, Lisp o Eiffel, i cui compilatori sono forniti da altri produttori.
Il Common Language Runtime invece è il motore d'esecuzione della piattaforma .NET, esegue cioè codice CIL (Common Intermediate Language) compilato da compilatori che possono avere come target il CLR. Questo componente si occupa di compilare just-in-time (al volo) il codice CIL in linguaggio macchina comprensibile alla CPU.
Microsoft iniziò lo sviluppo della tecnologia .NET verso la fine degli anni novanta, sotto il nome di Next Generation Windows Services (NGWS). La prima beta del Framework .NET fu rilasciata verso la fine del 2000. Il framework è stato quindi aggiornato più volte nel corso degli anni. La prima versione ufficiale fu la 1.0 e venne rilasciata da Microsoft nei primi mesi del 2002 parallelamente all'uscita dell'ambiente di sviluppo Visual Studio.Net. Negli anni a seguire .NET ha subito numerose modifiche atte a ottimizzarne il funzionamento ed espanderne

le potenzialità ed attualmente è arrivato alla sesta versione, la 4.0, uscita nell'aprile 2010 in concomitanza con la commercializzazione dell'ultima edizione dell'ambiente di sviluppo, ovvero Visual Studio 2010.

3.2.1.1 Il C.L.R.

Il principale strumento su cui ruota il .NET è il Common Language Runtime: è un livello del framework posto sopra il sistema operativo che gestisce l'esecuzione delle applicazioni .NET. I programmi scritti in .NET non comunicano direttamente con il sistema operativo ma attraverso il CLR. Esso è quindi responsabile dell'esecuzione vera e propria delle applicazioni, assicurando che vengano rispettate tutte le dipendenze e gestendo la memoria, la sicurezza, l'integrazione del linguaggio e tutti gli altri aspetti relativi al funzionamento di tali software.

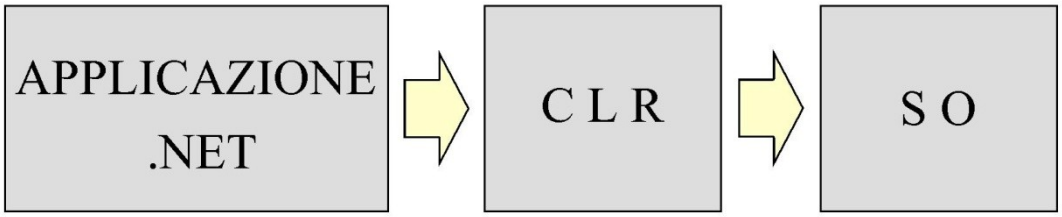

Figura 9: Schema funzionamento software sviluppati in .NET

Il CLR, chiamato anche brevemente runtime, fornisce numerosi servizi che consentono di semplificare la stesura del codice, la distribuzione dell'applicazione e di migliorare l'affidabilità della stessa.
Il codice a cui il runtime fa riferimento e la cui esecuzione è gestita da esso, è detto codice gestito (managed code); codice macchina non sicuro (unmanaged code), perché aggira il runtime, può invece essere generato dai compilatori. L'accesso alle Garbage Collection è possibile solo attraverso il codice gestito. La responsabilità per attività quali la creazione di oggetti, l'esecuzione di chiamate a metodi e cosi via, è richiesta al Common Language Runtime che consente di fornire servizi aggiuntivi al codice in esecuzione. Il codice gestito ha accesso al Common Language Runtime attraverso il quale può avvantaggiarsi delle caratteristiche della piattaforma (integrazione multi-linguaggio, gestione delle eccezioni, sicurezza, gestione delle versioni, ecc.).

Il Common Language Runtime è composto da cinque componenti che sono:

• CTS – Common Type System

Il CLR utilizza un sistema di tipi unificato in grado di esprimere la semantica dei moderni linguaggi di programmazione. Tale sistema definisce un insieme standard di tipi di dato e di regole necessarie per la realizzazione di nuovi tipi i quali sono realizzabili ed eseguibili dal runtime. I compilatori del .NET Framework utilizzano i servizi del runtime per definire i tipi di dato, gestire gli oggetti ed eseguire chiamate a metodi invece di utilizzare i metodi specifici dello strumento o del linguaggio. L'utilizzo di un sistema di tipi unificato offre come risultato una profonda integrazione tra i linguaggi. Il codice scritto in un linguaggio può ereditare l'implementazione da classi scritte in un altro linguaggio; le eccezioni possono essere sollevate dal codice scritto in un linguaggio e gestite da codice scritto in un altro, e operazioni come il debugging e profiling operano in modo trasparente indipendentemente dal linguaggio utilizzato per realizzare il codice. Ciò significa che non è più necessario creare versioni differenti della stessa libreria per ogni linguaggio di programmazione o compilatore, e che, per quanto riguarda le librerie di classe, non si è più limitati a quelle sviluppate per il linguaggio di programmazione utilizzato.

• CLS – Common Language Specification

Il CLS è un sottoinsieme del Common Type System formato da una serie di regole che vengono utilizzate per generare gli assembly e alle quali tutti i fornitori di librerie di classi e progettisti di linguaggi, che mirano a sfruttare il CLR, devono sottostare. Se un componente scritto in un linguaggio dovrà essere utilizzato da un altro linguaggio allora chi realizza il componente dovrà aderire ai tipi e alle strutture definite dal CLS.

Le librerie realizzate con codice che incarna il CLS vengono chiamate CLS framework e devono adeguarsi ad una serie di regole, tra le quali:

- banalmente evitare l'uso di nomi utilizzati comunemente come parole chiave nei linguaggi di programmazione;
- non permettere all'utente di costruire tipi nidificati;
- assumere che le implementazioni dei metodi con lo stesso nome e firme in differenti interfacce siano indipendenti.

• CIL – Common Intermediate Language

L'implementazione nel .NET Framework del Common Intermediate Language è chiamata Microsoft Intermediate Language (MSIL). Tutti i compilatori che si uniformano alla struttura del CLR devono generare una rappresentazione intermedia del codice, indipendente dall'hardware della piattaforma in uso, chiamata Common Intermediate Language. Il runtime utilizza questo linguaggio intermedio per generare codice nativo oppure viene eseguito al volo mediante la compilazione JIT (just in time). Il CIL si pone a un livello molto più alto della maggior parte dei linguaggi macchina, avendo istruzioni per il caricamento, la memorizzazione e l'inizializzazione dei dati, per richiamare metodi da oggetti e molte istruzioni di tipo convenzionale per le operazioni aritmetiche e logiche, il controllo di flusso, l'accesso diretto alla memoria o per generare ed intercettare le eccezioni per la gestione degli errori. Questo formato intermedio presenta contemporaneamente affinità e grandi differenze rispetto al tradizionale linguaggio Assembly, similmente al quale può operare sui dati con istruzioni di tipo "push" e "pop" e spostarli nei registri; diversamente dallo stesso non fa però riferimento a nessuna particolare piattaforma hardware. Uno dei principali vantaggi di questa soluzione è che permette al CLR di verificare durante la compilazione che il codice gestito sia completamente conforme alle specifiche di programmazione e ai tipi di dati previsti dal runtime. Durante questa verifica il CLR controlla ad esempio l'uso corretto dei puntatori e si assicura che non siano presenti conversioni tra tipi non consentite prima che il codice gestito sia eseguito. L'IL è convertito al volo in codice specifico per la CPU da un compilatore JIT, oppure compilato in codice nativo durante l'installazione. Il runtime fornisce uno o più compilatori a seconda del numero di piattaforme che deve supportare: nonostante l'attuale limitazione ad ambienti Microsoft, questo garantisce una buona indipendenza. Quando un compilatore conforme al CLS genera il linguaggio intermedio, produce anche metadati che descrivono i tipi specifici appartenenti al Common Language Types (CLT) utilizzati nel codice, comprendente la definizione di ogni tipo, le firme per ogni membro del tipo, i membri ai quali il codice fa riferimento e gli altri dati che il runtime usa durante l'esecuzione. Il MSIL e i metadati sono contenuti in un file Portable Executable (PE), un'estensione del formato Microsoft Portable Executable e simile al Common Object File Format utilizzato nel mondo Unix per gli eseguibili. All'utente i PE appaiono come familiari file .DLL e .EXE. Il formato dei file

permette di ospitare sia il codice IL che il codice nativo, i metadati e un “pattern signature” che permette al sistema operativo di riconoscere le “immagini” (nel senso di unica porzione contigua di codice) del Common Language Runtime. La presenza dei metadati nei file eseguibili permette ai componenti di essere autodescrittivi, eliminando di fatto la necessità di librerie dei tipi aggiuntivi o del tipo Interface Definition Language (IDL) usate in DCOM e CORBA. Il runtime localizza ed estrae i metadati del file quando è necessario durante l'esecuzione del codice.

• JIT – Just In Time Compiler

Prima che l'Intermediate Language possa essere eseguito deve essere convertito dal compilatore Just In Time di .NET Framework in codice nativo, che è specifico della CPU e funziona sulla stessa architettura sulla quale il compilatore JIT stesso sta funzionando.

I progettisti Microsoft insistono sul fatto che il runtime non interpreta mai nessun linguaggio, ma esegue sempre la conversione e l'esecuzione di codice nativo, persino per i linguaggi di script come VBScript, con evidente vantaggio sulle prestazioni. Il principio di funzionamento che è dietro i compilatori JIT è quello per il quale alcune parti di codice di un programma possono non essere mai chiamate in causa durante l'esecuzione di un programma; quindi piuttosto che sprecare tempo e memoria per convertire tutto il CIL di un file Portable Executable in codice nativo, il JIT converte l'IL solamente quando è necessario e ne memorizza il codice nativo risultante per renderlo disponibile per le chiamate successive. Il loader crea e allega uno stub, un componente software necessario ad eseguire una Remote Procedure Call (RPC), ad ogni metodo del tipo quando questo viene caricato; alla chiamata iniziale del metodo, lo stub passa il controllo al compilatore JIT il quale converte l'IL di quel metodo in codice nativo e modifica lo stub per dirigere l'esecuzione alla locazione del codice nativo. Le chiamate successive al metodo già compilato dal JIT procedono direttamente verso il codice nativo generato precedentemente, riducendo il tempo necessario alle successive compilazioni ed esecuzioni del programma da parte del compilatore JIT. Il codice cosi compilato deve sottoporsi a un processo di controllo in cui vengono esaminati l'Intermediate Language e i metadati per stabilire che essi siano sicuri ovvero che accedano esclusivamente alle locazioni di memoria autorizzate, che le identità siano verificate e che i riferimenti ai tipi di

dato siano compatibili con i tipi stessi. Questa caratteristica offre uno strato di protezione automatico dagli errori di programmazione.
Sono previsti due tipi di compilatori JIT, quello normale e la versione ridotta economy. Il compilatore normale esamina l'IL di un metodo e lo converte in codice nativo ottimizzato per la piattaforma esattamente come fa un tradizionale compilatore C/C++. Il compilatore economy invece, è stato progettato per l'utilizzo su quelle macchine per le quali il costo per l'uso della memoria e dei cicli di CPU è elevato. Il compilatore economy semplicemente rimpiazza ogni istruzione MISL con la sua controparte nativa. Questo tipo di compilazione è molto più rapida di quella tradizionale ma il codice prodotto è più inefficiente in quanto non viene ottimizzato per la specifica piattaforma. Questo tipo di codice risulta comunque più funzionale di quello interpretato. Il compilatore economy richiede meno memoria per funzionare, (quindi sarà preferito per i dispositivi portatili) grazie ad una maggiore semplicità progettuale e a meccanismi come il Code Pitching che permette al CLR di eliminare dalla memoria il codice nativo dei metodi non utilizzati. Se un metodo non viene utilizzato per un certo periodo di tempo, il runtime preleva il blocco di codice nativo completo la volta successiva che il metodo verrà invocato. Probabilmente a un confronto diretto il codice compilato Just In Time con il compilatore standard risulterà comunque più lento del tradizionale codice non gestito, proveniente ad esempio da un compilatore C/C++. Tuttavia per sua stessa natura il compilatore Just In Time lavora direttamente sulla macchina di destinazione del programma e pochi istanti prima di eseguire il programma stesso. Questo fornisce al JIT dati sull'ambiente di destinazione che nessun compilatore tradizionale potrà mai avere permettendo un elevato grado di ottimizzazione. Anche l'esatta conoscenza dello stato della memoria e dei registri dell'ambiente di esecuzione può rappresentare un dato rilevante per eseguire un'ulteriore ottimizzazione del codice.

• VES – Virtual Execution System
Rappresenta l'equivalente della macchina virtuale Java per l'ambiente di Sun/Oracle. Il VES carica, realizza i collegamenti ed esegue i programmi scritti per il Common Language Runtime. Il VES adempie le sue funzioni di loader utilizzando le informazioni contenute nei metadati ed utilizza il late binding per integrare moduli compilati separatamente, che possono essere anche scritti in linguaggi differenti. Il VES inoltre fornisce servizi durante l'esecuzione dei codici,

che includono la gestione automatica della memoria, supporto per profiling e debugging, sandbox per la sicurezza analoghe a quelle Java e l'interoperabilità con il codice non gestito come ad esempio componenti COM.
Come detto, quando viene compilato un programma .NET scritto in un qualsiasi linguaggio .NET (C#, VB.NET), il codice sorgente non viene direttamente tradotto in codice eseguibile binario ma in un codice intermedio, chiamato MSIL (Microsoft Intermediate Language), il quale viene poi interpretato dal CLR. Questo linguaggio intermedio è indipendente dall'hardware e dal sistema operativo. Solo in fase di esecuzione il CLR si occupa di tradurre il codice MSIL in codice eseguibile binario.

3.2.2 VISUAL STUDIO .NET

Figura 10: logo Visual Studio 2008

Visual Studio è un ambiente di sviluppo integrato (Integrated development environment o IDE) sviluppato da Microsoft, che supporta attualmente diversi tipi di linguaggio, quali C, C++, C#, F#, Visual Basic .Net e ASP .Net, e che permette la realizzazione di applicazioni desktop e di siti, applicazioni e servizi web. È inoltre un RAD (Rapid Application Development), ovvero una applicazione creata per velocizzare la produzione di software e aumentare la produttività del programmatore con mezzi come l'intellisense (Figura 11) o un designer visuale delle forms. IntelliSense è una forma di completamento automatico che serve, oltre a questo scopo, come documentazione per i nomi delle variabili, delle funzioni e dei metodi usando metadati e reflection.

Integrando questa tecnologia Visual Studio è in grado di correggere eventuali errori sintattici (ed alcuni logici) senza compilare l'applicazione, e possiede, oltre che a diversi strumenti per l'analisi prestazionale, un potente debugger interno per il rilevamento e la correzione degli errori logici nel codice durante il runtime.
Visual Studio è oltretutto multipiattaforma: con esso è possibile realizzare programmi per server, workstation, pocket PC, smartphone e, naturalmente, per i browser.
La scelta di utilizzare questo IDE da parte del team coinvolto nel progetto è stata dettata dal fatto che questo prodotto Microsoft è ottimizzato per la tecnologia ed i linguaggi .NET essendo nato e cresciuto parallelamente ad essi, inoltre va sottolineato che Visual Studio è un software per lo sviluppo davvero completo ed esauriente.

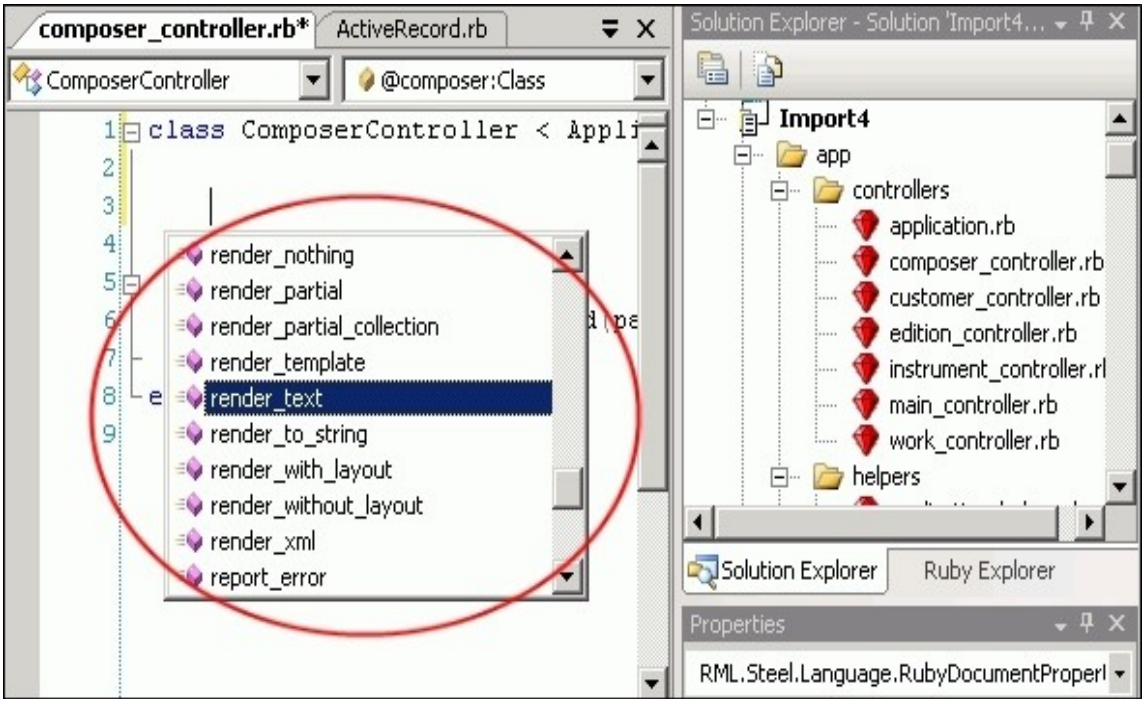

Figura 11: esempio di intellisense

3.2.2.1 VISUAL BASIC .NET

Visual Basic, chiamato precedentemente Visual Basic .NET, è il successore del vecchio Visual Basic, il quale era arrivato fino alla versione 6, ed è uno dei linguaggi principali di Visual Studio, la suite per programmatori di Microsoft,

basata interamente sul Framework .NET.. Le principali novità di questo nuovo linguaggio di sviluppo del Framework .NET sono che esso è basato su oggetti, implementa infatti completamente la programmazione orientata agli oggetti, e che è compilato.
Il Visual Basic, e il suo corrispondente C#, sono un punto di congiunzione di tante iniziative e tecnologie di sviluppo Microsoft del passato come ad esempio: il COM e gli ActiveX, sostituiti con assembly e classi di oggetti completamente gestiti: per gestiti si intende che la gestione della memoria è gestita autonomamente dal garbage collector. Con Visual Basic è possibile realizzare applicazioni windows forms, web, servizi, componenti COM, MQ, Web service ed anche destinate a dispositivi mobile tramite l'uso delle librerie del Compact framework.
Dal 2002, anno della prima uscita del software, fino al 2010 si sono susseguiti cinque rilasci di nuove versioni dello stesso. L'evoluzione nel corso di questo periodo è stata contrassegnata solamente dall'aggiunta progressiva di nuove funzionalità, di nuove caratteristiche e metodologie per la programmazione, di miglioramenti alle librerie e di molti altri accorgimenti; tutto ciò per poter in questo modo modernizzare il linguaggio e tenerlo al passo con le costanti innovazioni della programmazione odierna.

Figura 12: logo Visual Basic. NET

3.2.2.1.1 DIFFERENZE CON VB 6

Va fatto notare che Visual Basic.Net non è una semplice evoluzione di VB6, ma è a tutti gli effetti un altro linguaggio, adatto alla nuova generazione di software e di sistemi operativi. Benché all'inizio si pensasse ad un semplice aggiornamento del linguaggio, in realtà VB.NET e VB non sono compatibili. Nell'IDE di sviluppo è infatti integrato un convertitore, che però garantisce una conversione solo parziale del codice, obbligando ad una totale revisione dello stesso, in virtù del fatto che comunque la tecnica di programmazione è stata completamente stravolta perché passata da imperativa (anche se basata su eventi) ad orientata agli oggetti.
Ciò costituisce un'evoluzione della strategia Microsoft che in passato aveva tradizionalmente avuto nel linguaggio Visual Basic uno dei propri punti di forza. Alcuni analisti hanno fatto notare che si è trattato di una scelta non priva di rischi da parte di Microsoft, in quanto per un programmatore Visual Basic la migrazione verso Visual Basic .NET può richiedere un notevole periodo di apprendimento, poiché, per poter sfruttare tutte le potenzialità del nuovo linguaggio, deve abituarsi a pensare in termini totalmente object oriented, e questo nonostante il fatto che Visual Basic .NET abbia conservato, per quanto possibile, la sintassi delle vecchie versioni.

3.2.2.2 VISUAL C .NET

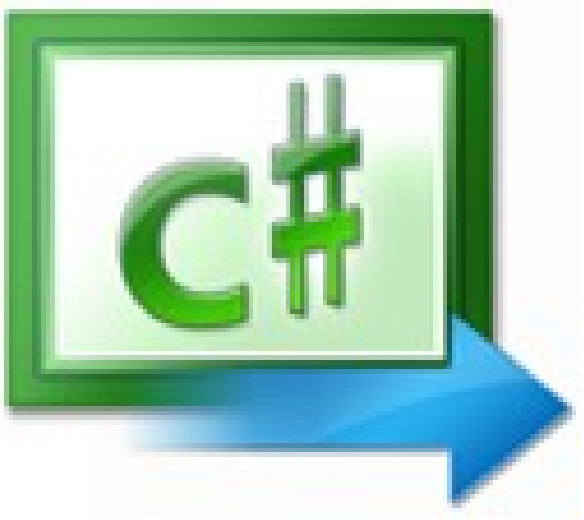

Figura 13: logo Visual C .NET

Microsoft Visual C, noto anche come MSVC, è un ambiente di sviluppo integrato (IDE) di Microsoft per la programmazione nei linguaggi C, C++ e C++/CLI.
È orientato soprattutto allo sviluppo e al debug di codice C++ basato sulle API di Microsoft Windows, DirectX e Microsoft .NET.
Esiste sia in una versione stand-alone, ovvero Microsoft Visual C++ 2008 Express Edition, che come parte dell'ambiente Microsoft Visual Studio nelle tre versioni Standard, Professional e Team Suite. Il sito di MSDN ne mette a disposizione la versione "Express" per il download gratuito.

3.3 TORTOISE SVN

Figura 14: logo Tortoise SVN

Tortoise SVN è un client grafico subversion scritto per funzionare come un'estensione di Microsoft Windows ed è inoltre un programma gratuito rilasciato sotto licenza GNU GPL (General PublicLicense).
Subversion è un sistema di controllo versione di software, dati, documenti o anche foto e file multimediali. Il sistema appunto, a differenza di CVS (Concurrent Versioning System), supporta ogni tipo di file che abbiamo nel nostro computer. Per fare un esempio pratico e veloce di quali siano le potenziali operazioni che un software di questo tipo può eseguire si supponga di avere un documento Word che viene aggiornato con una certa frequenza oppure un software o un programma in php eseguibile su server. Grazie al software per il controllo di versione tra la lunga lista di cose possibili si può:

- tener traccia di tutte le modifiche effettuate su un file;

- aprire un file in modifica e bloccarlo ad eventuali modifiche di altre persone fino a quando non avremo terminato il nostro lavoro;
- in caso di propagazione di errori bloccarli tranquillamente e in qualsiasi momento ritornare ad una versione precedente;
- sviluppare versioni parallele di un software;
- eseguire backup in qualsiasi momento, di tutte le versioni di un programma in fase di sviluppo.

La particolarità di TortoiseSVN inoltre, non è solo quella di togliere l'impiccio dei comandi da terminale, che a volte possono risultare complessi, ma di fornire un'interfaccia che consente di sfruttare ogni funzione operabile all'interno di un repository locale.
Quindi non è solo un tool destinato all'utilizzo di chi vuole avvicinarsi a questo sistema di gestione codice, ma anche per chi semplicemente vuole un modo più comodo e immediato per gestire le sue operazioni giornaliere.
Anche l'installazione, diversamente da quanto possa sembrare, non è molto intrusiva: tutto ciò che viene modificato è che si avrà una nuova opzione "TortoiseSVN" in ogni context-menu (Figura 15).

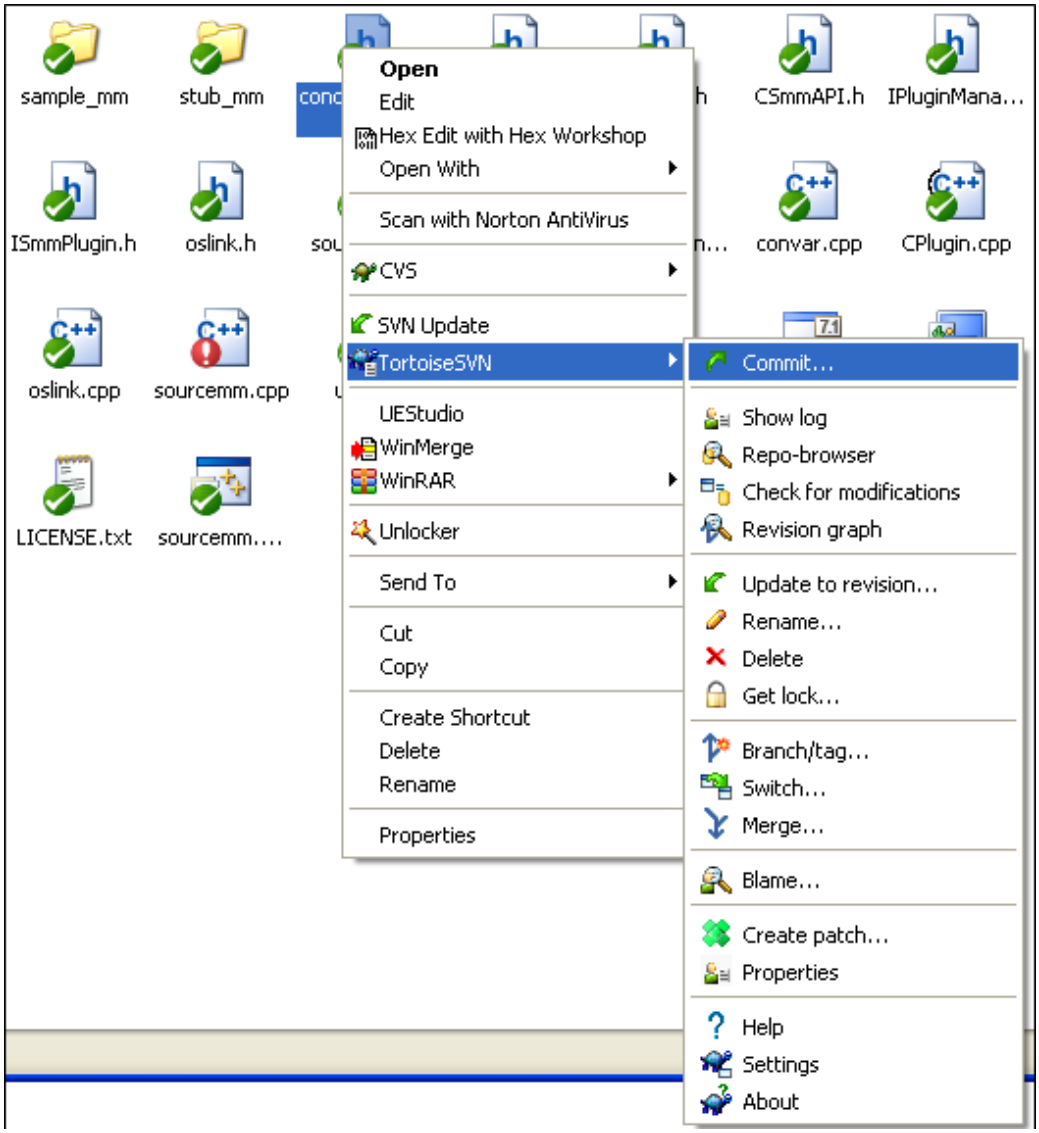

Figura 15: vista del menu tasto destro dopo l'installazione di Tortoise SVN e delle icone dei file ad esso sincronizzati

Una volta installato questo software e aggiornato tutte le impostazioni ed i settaggi utili per poter procedere con lo sviluppo del proprio software ogni volta si voglia effettuare un'operazione di aggiornamento o di verifica basta cliccare col pulsante destro sul file e vengono visualizzate le opzioni di gestione di Repository SubVersion. Inoltre con questa apllicazione viene installato anche un set di icone che permettono di vedere da subito in quali cartelle sono contenuti i Repository.

TortoiseSVN può inoltre integrarsi con Microsoft Visual Studio utilizzando i plugin di terze parti VisualSVN e VsTortoise ed assieme ad esso viene anche

distribuito TortoiseMerge uno strumento opensource utile per mostrare le differenze tra due file.
Nel 2007 TortoiseSVN ha vinto il premio di SourceForge.net per il più utile strumento per gli sviluppatori, votato dalla comunità.

3.4 XVI32

XVI32 è un editor esadecimale gratuito che funziona sotto Windows 9x/NT/2000/XP/Vista/7 il quale nome deriva da XVI, la notazione romana per indicare il numero 16 di quella decimale.
Un editor esadecimale, o hexeditor, è un'applicazione in grado di gestire la rappresentazione in formato esadecimale dei singoli byte di qualunque tipo di file, e di consentirne la modifica.
A differenza degli editor specializzati, ad esempio quelli di testo, che normalmente non visualizzano i caratteri di controllo, l'editor esadecimale non effettua alcuna distinzione.
La rappresentazione in formato esadecimale è particolarmente efficiente, poiché ogni singolo byte viene sempre rappresentato da una coppia di caratteri, tuttavia sono spesso disponibili formati di visualizzazione alternativi, quali il decimale o l'ottale. Analogamente, i soli byte coincidenti con un carattere visualizzabile secondo la codifica ASCII possono venir rappresentati con il relativo carattere tipografico.
Un software con questo tipo di funzionalità può essere utilizzato per vari scopi: si possono apportare modifiche ai programmi eseguibili, per esempio traducendo le stringhe da una linguaggio ad un altro, oppure si possono modificare piccole parti di codice. Inoltre, gli utilizzatori più esperti possono servirsene per recuperare dati da file corrotti che non vengono più aperti dall'applicazione registrata.
Nel caso del progetto in questione questo software è stato utilizzato per eseguire i controlli sullo scambio di pacchetti che avvenivano tra il Programmer e le centraline antintrusione, il primo infatti una volta impostata e iniziata la connessione crea un file .BIN dove vengono registrati tutti i messaggi scambiati.

La schermata di un comune hexedit è divisa in tre colonne: la posizione sul file (offset o indirizzo), la visualizzazione esadecimale e la visualizzazione ASCII degli stessi dati.

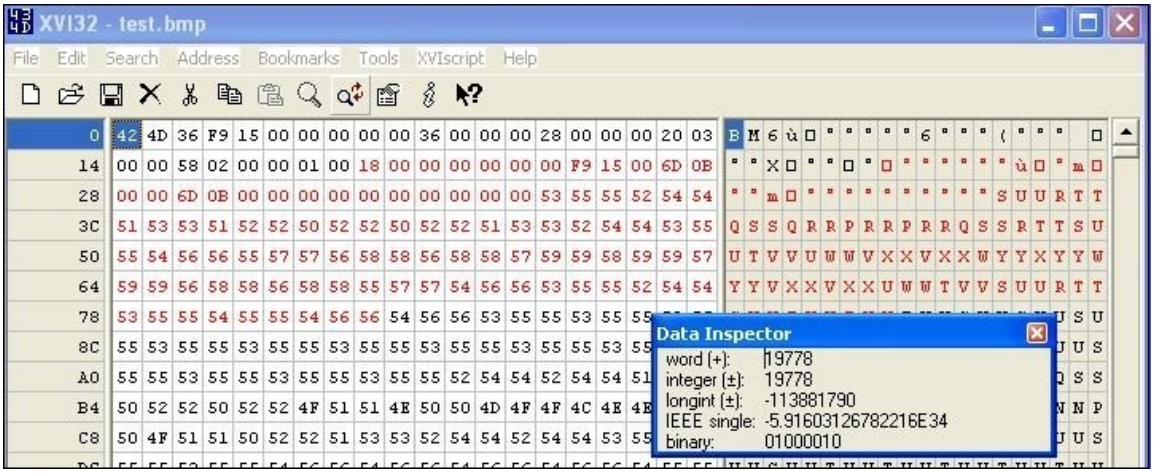

Figura 16: interfaccia grafica XVI32

3.5 CSDIFF

CSDiff è un applicativo avanzato e gratuito per l'analisi e la rilevazione delle differenze. Installabile su piattaforma Microsoft Windows, è una grande risorsa per chiunque abbia bisogno di analizzare i cambiamenti che sono stati fatti tra due revisioni dello stesso file o del contenuto di una cartella.

CSDiff è comunemente usato quando si analizzano file sorgenti di software, documenti HTML e MS-Word. Una persona che ha a che fare abitualmente con lo sviluppo o la modifica di software incappa spesso nel problema di dover ripristinare una versione di un sorgente precedente per annullare le modifiche effettuate nel caso queste abbiano comportato l'introduzione di errori o il malfunzionamento del programma in fase di elaborazione. Questo applicativo permette allo sviluppatore di confrontare le due versioni del codice e poter così determinare le variazioni introdotte ed annullare le modifiche incriminate. Inoltre, può essere altrettanto frequente dover analizzare quali sono i file di un progetto che sono stati modificati dopo una determinata versione dello stesso.

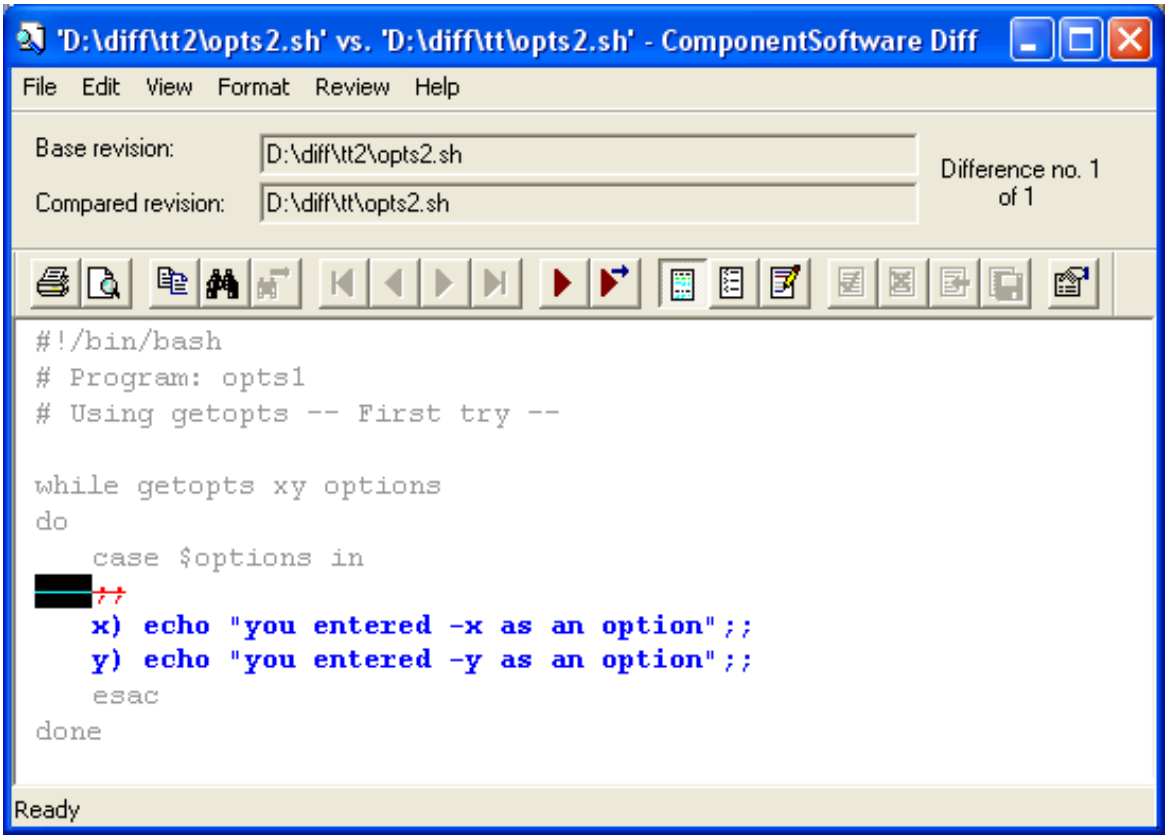

Figura 17: interfaccia grafica CSDIFF

Alcune tra le caratteristiche principali di CSDiff sono le segenti:

- la modalità di visualizzazione a singolo display fornisce un unico ed intuitivo stile per l'analisi delle differenze e lo rende ideale per la stampa e i reports;
- in modalità editabile sono permesse le eventuali modifiche manuali delle parti di codice che si intende variare;
- si può decidere se rilevare i cambiamenti per riga, per parola o per singolo carattere;

4 REALIZZAZIONE

4.1 PREPARAZIONE

Come pianificato al momento dell'inserimento nel progetto è stato necessario iniziare il tirocinio cercando di apprendere più informazioni possibili sul software Programmer Sicurezza e sulla famiglia di centrali antintrusione interessate dal programma di aggiornamento. Si è pensato quindi di agire gradatamente andando a studiare tutti i componenti ed eseguendo infine dei test sperimentali sul loro funzionamento.
Il primo passaggio è consistito nella lettura del Manuale Installatore per centraline di tipo Defender 8-12 (copertina in Figura 18). Durante la visione ed analisi di questo documento si è cercato di comprendere la struttura di questo tipo di dispositivi e i vari sotto componenti da cui sono formati, inoltre si è approfondito in modo particolare l'aspetto della connessione tra centrali e PC in modo da comprendere le modalità con cui i due elementi interagiscono.

Figura 18: Copertina manuale installatore per Defender 8-12

Dopo questa prima parte è stato necessario analizzare il funzionamento del software Programmer Sicurezza. Una volta installato questo programma sul PC in dotazione ne è stata studiata inizialmente l'architettura grafica e poi le varie

funzionalità e impostazioni modificabili, andando ad esplorare e testare tutti gli elementi presenti nell'interfaccia.
Una volta compresi tutti i particolari del funzionamento del Programmer è iniziata la fase di test fisici attraverso l'uso di un esemplare di Defender. Nella prima parte dei test si è cercato di allestire un ipotetico impianto antintrusione utilizzando campioni dei vari componenti disponibili nel laboratorio quali tastiere di comando, combinatori telefonici, un circuito stampato simulante le linee di allarme e altri elementi che ampliano le caratteristiche dell'impianto. Di seguito sono stati effettuati dei test sugli allarmi e le altre funzionalità supportate, come ad esempio le chiamate, il controllo via toni o l'uso di chiavi elettroniche per l'attivazione o la disattivazione dell'impianto completo o di aree singole.
Infine l'ultimo step è stato quello di effettuare le prove di connessione tra PC e centrale. In questo passaggio si sono approfondite le procedure di upload delle configurazioni impostate tramite Programmer e il download di quelle memorizzate nella centrale, andando a verificare che le modifiche effettuate venissero ogni volta impostate correttamente.
In questa fase di training iniziale è stata importante la figura dell'Ing. Fusaro, che con la sua pazienza e disponibilità è riuscito a dissolvere eventuali dubbi o incomprensioni che interessavano il funzionamento di tali dispositivi mediante descrizioni accurate ed esempi pratici.
Altro passaggio fondamentale è stato quello di imparare ad utilizzare il software di programmazione e la parte d'interfaccia dell'IDE fornito da Visual studio. Per fare ciò è stato necessario leggere delle opportune guide presenti nel web oppure scaricabili gratuitamente dal sito http://msdn.microsoft.com. Ovviamente questa sessione di studio ha fornito solamente i fondamentali necessari al primo approccio con la suite e la maggior parte delle nozioni sono state apprese durante la successiva fase di debug in cui si sono dovuti studiare gli errori e progettare le relative correzioni.
Una volta completata l'ottimizzazione della parte d'interfaccia del software, sia dal punto di vista grafico che da quello della stabilità, è stata necessario studiare in modo più approfondito le caratteristiche del funzionamento dell'iterazione PC-centrale. Prima infatti di iniziare la fase di test e di ricerca e correzione dei bugs presenti nelle porzioni di codice necessarie al suo funzionamento bisognava comprendere il meccanismo che stava alla base della comunicazione, ovvero il protocollo adottato per far interagire i due componenti.

4.1.1 IL PROTOCOLLO DI CONNESSIONE

L'interfaccia di collegamento adottato dal team di sviluppo per far interagire i vari modelli di centrale con una stazione host esterna è un protocollo progettato e realizzato ad hoc internamente all'azienda. Un manuale specifico ne spiega il funzionamento e ne descrive uno ad uno tutti i messaggi che compongono l'insieme di pacchetti interscambiabili tra le due unità.
La comunicazione tra la centrale e la stazione esterna avviene in maniera diretta, per quanto riguarda la porta seriale e il modem presenti nelle centraline, oppure può avvalersi di appositi moduli che permettono la comunicazione attraverso altri mezzi trasmissivi (ad esempio TCP/IP).
I sistemi di centraline antintrusione prese in considerazione in questa relazione permettono di definire diversi tipi di utenti, ciascuno con caratteristiche diverse: l'Installatore, che ha il completo controllo della configurazione della centrale; l'Amministratore, che può controllare tutta la centrale; il Master, che può definire nuovi utenti e gestire completamente l'intera area di sua competenza; gli Utenti Modificabili, definiti dal master, che possono avere il controllo parziale o totale dell'area a cui appartengono.
In base al tipo di utente deve essere possibile una diversa gestione della centrale, di conseguenza l'interfaccia di tipo "Clonazione di una Tastiera" (figura 19) sarà accessibile a tutti i tipi di utente mentre, il "Monitoraggio della Centrale" è ad appannaggio dei Master, dell'Amministratore e degli Utenti Modificabili. L'interfaccia di "Configurazione della Centrale" (figura 20) non è accessibile solo dagli utenti modificabili, e per questo alcuni comandi sono riservati ai Master e all'Amministratore, mentre molti altri sono dedicati all'Installatore.

Figura 19: Programmer in modalità di funzionamento " Clonazione di una Tastiera"

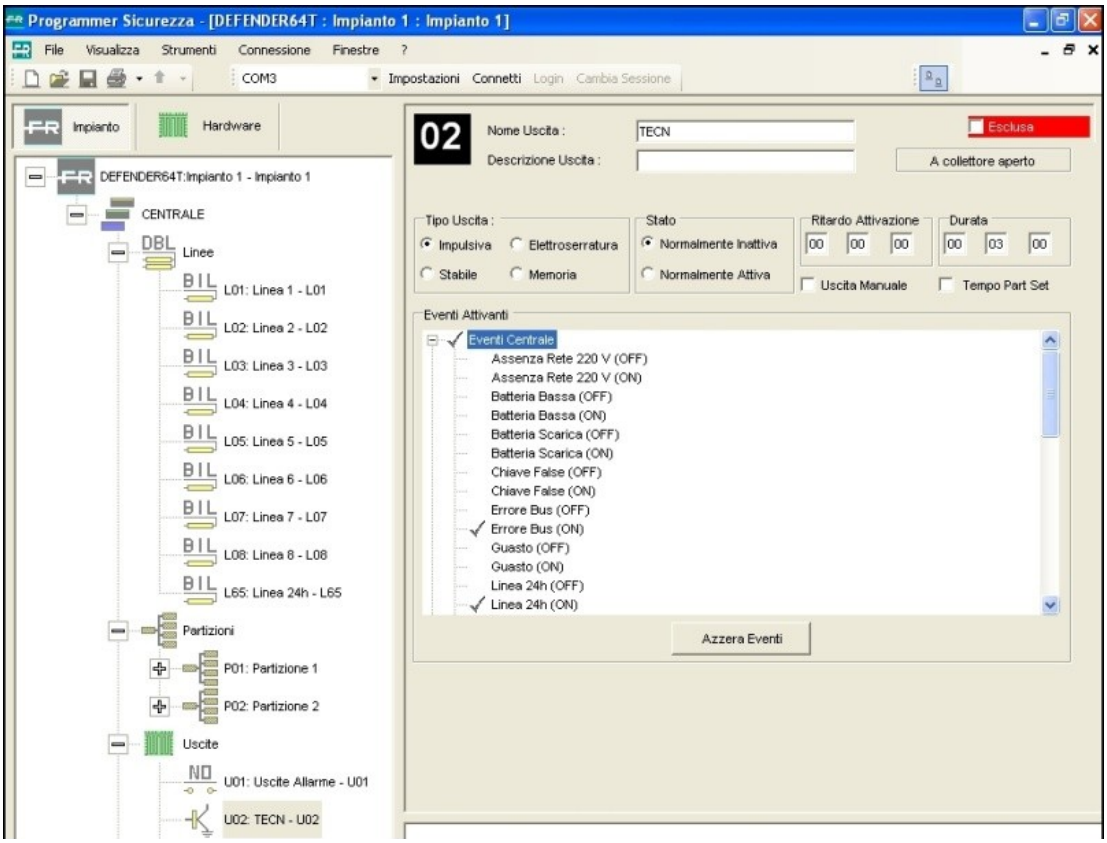

Figura 20: Programmer in modalità di funzionamento "Configurazione centrale"

Nel momento in cui si fa partire la connessione vi è dunque una prima fase necessaria, in cui i pacchetti del protocollo sono comuni a tutti gli utilizzatori delle centrali, ovvero la connessione ed autenticazione, dopodiché vi sono i comandi per la configurazione della centrale e poi per la clonazione della tastiera e per il monitoraggio della centrale.
Poiché la connessione può avvenire sia in locale che in remoto, e permette il completo controllo della centrale, si presta particolare attenzione alla modalità di connessione e, in particolare, alla sicurezza dei dati scambiati.
Le entità costituenti la rete in questione sono prive di qualsiasi funzione di "carrier sensing". Per semplificare il software di gestione del protocollo di accesso al canale è stata quindi adottata una configurazione di tipo Master-Slave half duplex.
L'unità host esterna funge da controllore e la centrale da periferica. In qualsiasi istante, la stazione esterna può fare richiesta di connessione. In base al canale utilizzato (porta seriale o modem; figura 21) la centrale si comporta in differenti modi:

- Connessioni tramite canale seriale (locale): la richiesta di connessione rimane attiva e inizia subito la comunicazione tra i due dispositivi;
- Connessioni tramite canale modem (remota) con call-back da centrale: la connessione viene interrotta e la centrale attiva una nuova connessione verso un numero telefonico prestabilito e memorizzato in centrale;
- Connessioni tramite canale modem (remota) senza call-back da centrale: la connessione non viene interrotta e può iniziare subito la comunicazione tra i due dispositivi;
- Connessioni tramite rete TCP/IP: è possibile, attraverso il dispositivo NetController (NTC), collegare la centrale attraverso la rete TCP/IP sfruttando le stesse funzionalità fornite con i tipi di connessione precedenti;

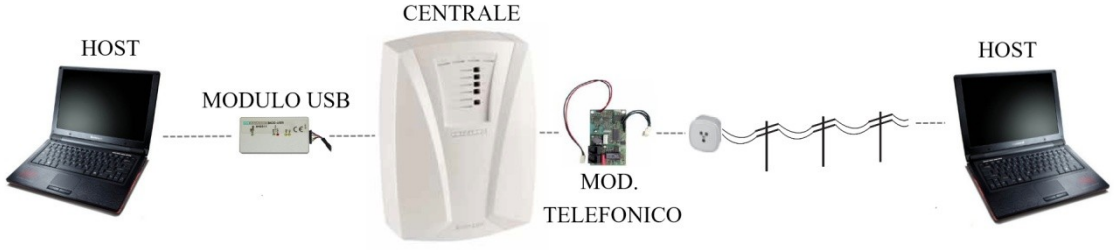

Figura 21: Schema di collegamento in locale e da remoto

In generale la comunicazione può essere sia di tipo “punto-punto” che di tipo “multi-punto”, dando quindi la possibilità alla stazione remota di gestire contemporaneamente più centrali. La velocità di comunicazione è di 2400 bps.

4.2 RICERCA BUGS

Nella prima fase del tirocinio, quella relativa alla ristrutturazione della parte d’interfaccia utente, è stato necessario effettuare una vera e propria sessione di ricerca degli errori critici sistematici e casuali che avvenivano con l’utilizzo del nuovo Programmer. La ricerca è stata eseguita cercando di utilizzare questa prima versione del nuovo software in tutti i suoi aspetti; questo ha significato l’uso intensivo del Programmer e la modifica di tutti i parametri e delle impostazioni presenti nell’interfaccia di configurazione di una centrale. Per cercare di ottimizzare la ricerca in termini di tempo e completezza è stato necessario mantenere un ordine logico e progressivo dei test andando a stimolare il programma settore dopo settore ed annotando tutti gli errori riscontrati.
Gli errori presenti erano classificabili in due macrocategorie: quella degli errori critici che causavano il crash del programma a seguito del lancio di un’eccezione non gestita (figura 22) e quelli di malfunzionamento, ovvero della risposta sbagliata del software ad una determinata serie di settaggi.

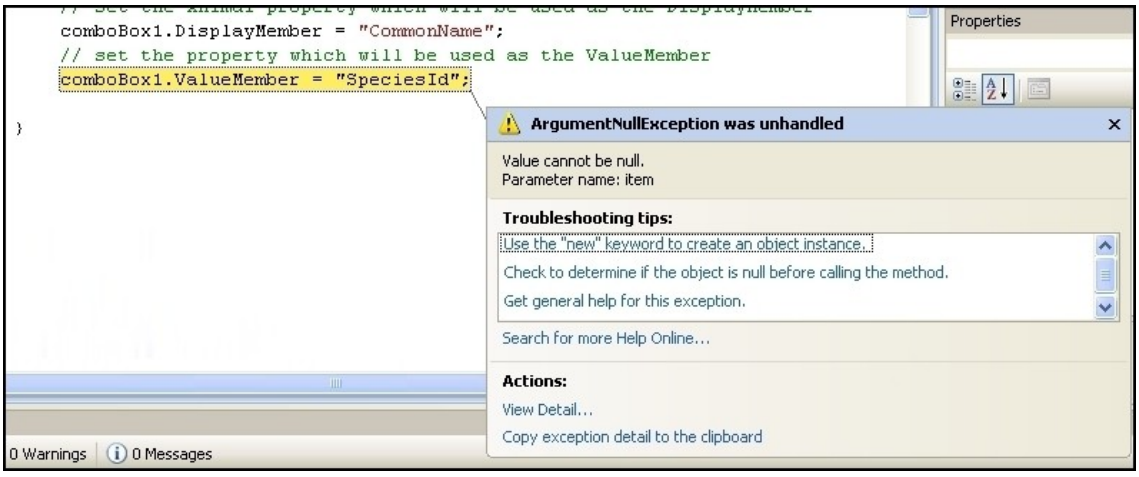

Figura 22: Eccezione non gestita

I primi causando appunto la terminazione del funzionamento del software costringevano al riavvio dello stesso e alla conseguente perdita di tempo, gli altri

invece, meno pesanti in termini di tempo, erano più difficili da diagnosticare perché non facili da notare per un utente non ancora esperto del software.
Oltre a questo tipo di problemi si è cercato di riscontrare anche tutte le imprecisioni ed i bugs grafici presenti in modo da poter in un secondo momento sistemare il nuovo programma anche da questo punto di vista. Si è trattato di verificare che le varie interfacce e schede presenti corrispondessero in tutti gli elementi a quelle del vecchio software e che inoltre vi fossero un minimo di ordine tra gli elementi e un posizionamento corretto degli stessi all'interno della videata.
Una volta che la ricerca ha iniziato a non dare più nuovi risultati si è passati alla fase di debug di tutti i problemi riscontrati e annotati.
Come già detto in precedenza l'intera esperienza del tirocinio si è articolata in più step e per questo vi sono state più fasi di ricerca dei bugs. La prima, appena trattata, ha comportato una lunga fase di debug con l'eliminazione errore dopo errore di tutti i problemi riscontrati. La seconda fase invece non è avvenuta interamente in un'unica sessione, questo perché gli errori che avvenivano durante i test sulla connessione tra centrale e pc, non permettevano un funzionamento corretto della stessa e quindi si sono dovuti eliminare da subito per poter così raggiungere passo dopo passo una connessione ottimale tra pc e centrale.

4.3 DEBUGGING

Dopo aver individuato gli errori presenti nel software si è dovuto affrontare la successiva fase di debugging cioè la rimozione di ogni singolo bug attraverso lo studio dello stesso e l'introduzione di una soluzione ottimale che garantisse un funzionamento corretto in linea con quello atteso.
Per studio dell'errore si intende la comprensione del tipo di errore e delle cause che portano ad esso. Il moderno IDE fornito da Visual Studio permette di accelerare i tempi in queste attività fornendo diversi strumenti per la visualizzazione delle variabili in tempo reale, gli stack delle chiamate, l'esecuzione passo passo di ogni singola istruzione e tante altre utilità. Una tra queste che merita di essere sottolineata è la generazione delle informazioni relative alle exception non gestite, al momento dell'esecuzione di un'istruzione critica che provoca lo stop del programma. Grazie ad essa si può risalire subito

alla riga di codice incriminata e capire immediatamente che tipo di errore ha causato l'impossibilità del software a proseguire con la successiva istruzione.
Ovviamente l'IDE viene incontro al programmatore cercando di fornire più informazioni possibili per individuare la posizione dell'errore, la tipologia dell'eccezione, lo stato delle variabili e tanti altri dettagli che aiutano a capire la causa del bug ma, non avendo intelligenza propria, non aiuta di certo lo sviluppatore a trovare una soluzione ad esso e deve quindi porvi rimedio da solo.
Rimediare il verificarsi di un bug consiste nel progettare una soluzione più o meno complessa per rendere funzionante il software. Al variare quindi del problema varia conseguentemente anche la tipologia e la complessità della soluzione.
Buona parte dei problemi riscontrati nella prima versione del nuovo Programmer erano dovuti principalmente alla conversione dal vecchio VB al nuovo VB.Net e quindi implicabili all'incompatibilità di alcune istruzioni o alle modifiche apportate alle caratteristiche di funzionamento di alcuni costrutti, i rimanenti invece consistevano in errori già presenti nel software precedente che, grazie alle regole poco ferree sul funzionamento di Visual Basic, permettevano al software di procedere ugualmente in modo corretto, o dandone l'impressione all'utente, e quindi emersi solo a seguito della conversione.

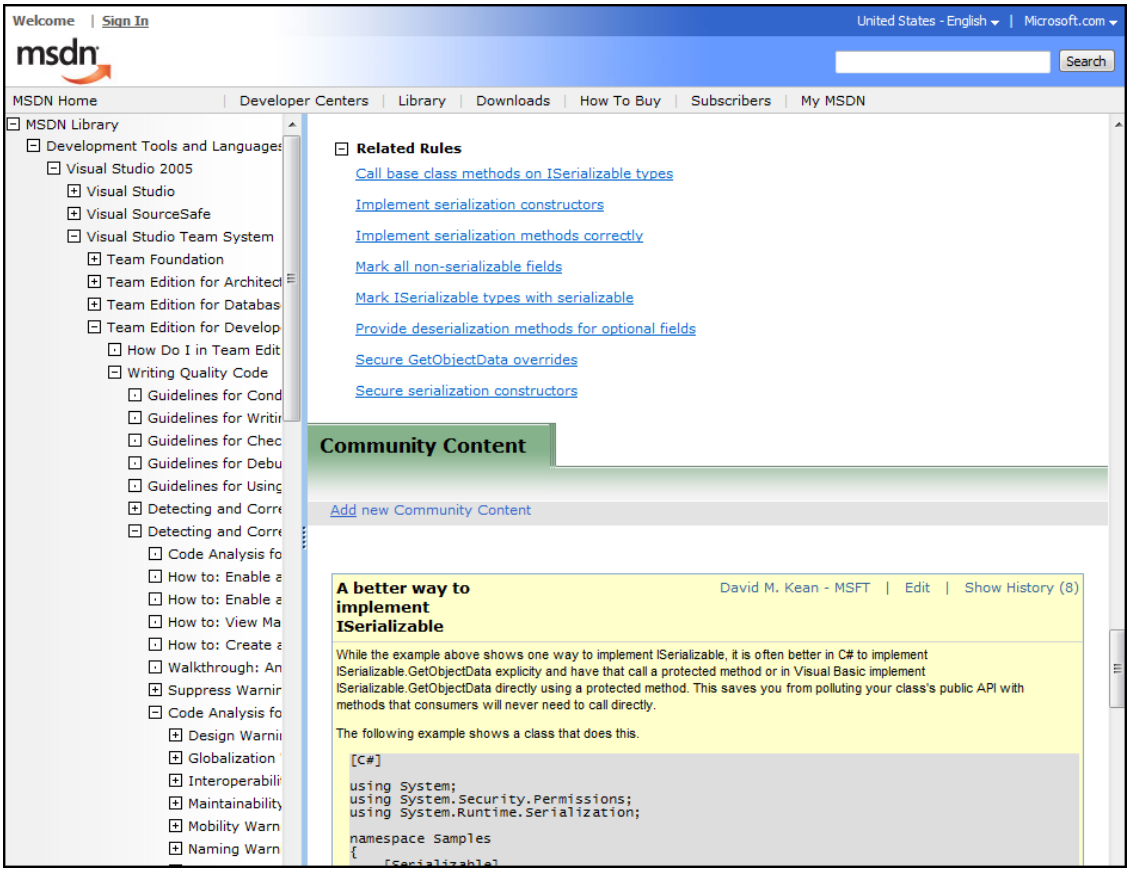

Figura 23: guida online Microsoft (MSDN)

Per quelli della prima tipologia è stato doveroso documentarsi mediante ricerche nel web in siti e forum relativi alla programmazione o nella guida online fornita da Microsoft (http://msdn.microsoft.com/it-it/; figura 23) per comprendere quali tipo di modifiche erano state effettuate ai costrutti, ai tipi di dati o alle classi di oggetti che creavano problemi nel nuovo software.

Le modifiche rilevate hanno comportato in seguito un adeguamento del software più o meno invasivo: in alcuni casi è stato necessario utilizzare funzioni diverse di una classe, mentre altre volte si è dovuto scrivere delle funzioni apposite per eseguire una attività che prima era fornita automaticamente da un metodo della classe, e di seguito viene fornito un esempio.

```
'=======================================================
' Name: SortTreeNodeCollection
' Input: ByVal tncParent As TreeNode; il nodo padre a cui  associata la collezione
         di nodi da ordinare
' Purpose: esegue l'ordinamento di una collezione di nodi associati ad un nodo padre
' Author: A.M.
'=======================================================

        Public Sub SortTreeNodeCollection(ByVal tncParent As TreeNode)

            Dim tnc(tncParent.Nodes.Count - 1) As TreeNode
            Dim tncString(tncParent.Nodes.Count - 1) As String
            Dim I, H As Integer
            Dim nodeTemp As TreeNode

            tncParent.Nodes.CopyTo(tnc, 0)
            tncParent.Nodes.Clear()
            For I = 0 To tnc.Length - 1
               tncString(I) = tnc(I).Name
            Next
            System.Array.Sort(tncString)
            For I = 0 To tnc.Length - 2
               For H = I To tnc.Length - 1
                  If tncString(I) = tnc(H).Name Then
                     If I = H Then Exit For
                     nodeTemp = tnc(I)
                     tnc(I) = tnc(H)
                     tnc(H) = nodeTemp
                     Exit For
                  End If
               Next
            Next
            tnc.Nodes.CopyTo(tncParent, 0)
```

Questa funzione permette di ordinare una collezione di nodi associata ad un unico nodo genitore, cosa che nel vecchio VB era eseguita automaticamente da un metodo della classe TreeNode.

Altri problemi riscontrati relativi al cambiamento del linguaggio di programmazione si sono avuti, dal punto di vista grafico, con la presenza di nuovi elementi a sostituzione o in aggiunta dei precedenti e con caratteristiche e particolarità diverse o supplementari.
Uno di questi è stata l'introduzione delle NumericUpDown (Figura 23) ovvero la fusione di una TextBox che può contenere solo valori numerici con un UpDownButton. Se da un lato questa introduzione agevola chi crea nuovo software fornendo con un unico elemento tutte le caratteristiche dei due precedenti, dall'altro chi adegua codice scritto in linguaggi antecedenti deve eliminarne le parti inutili ed adattarlo al nuovo componente, prestando molta attenzione al mantenimento del flusso di esecuzione voluto.

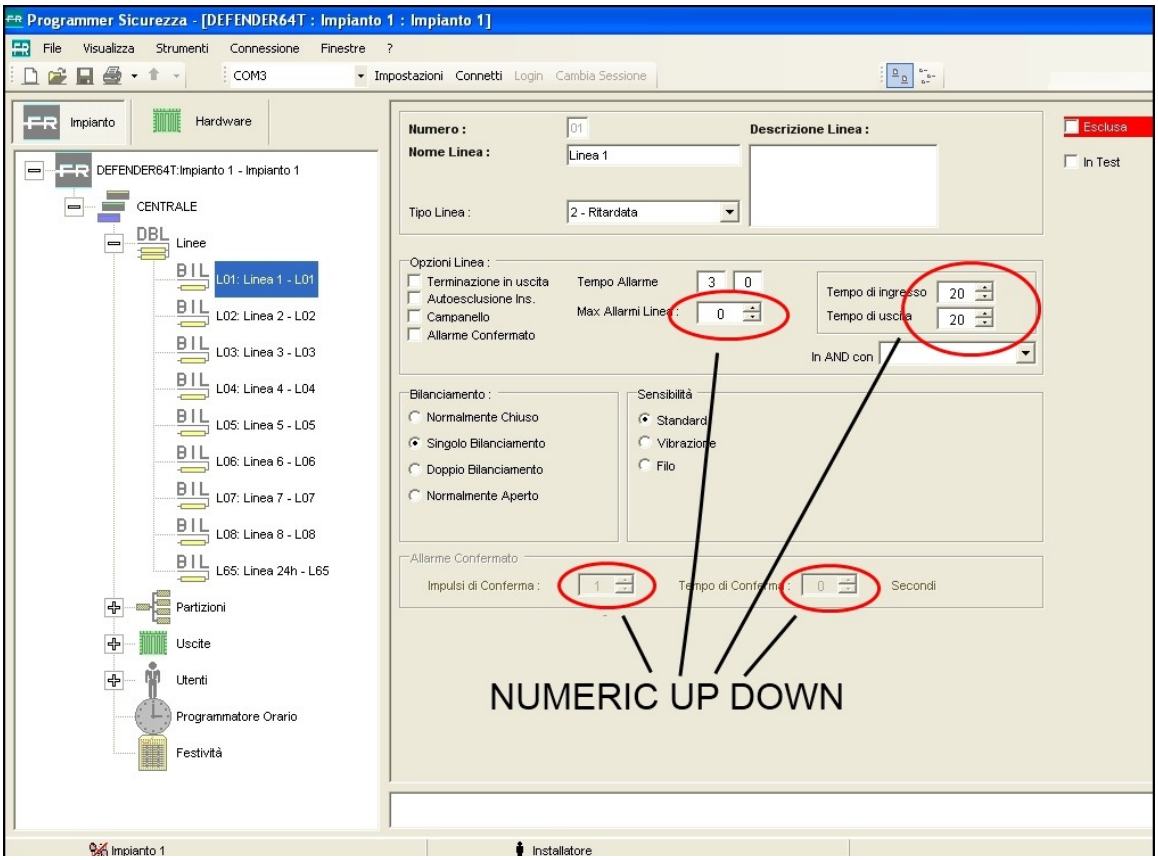

Figura 24: NumericUpDown

Ovviamente in un software che utilizza molti di questi componenti si dovrà inquadrare il lavoro su ciascun elemento preso singolarmente, e uno dopo l'altro aggiornarli tutti, è quindi ovvio che all'aumentare del numero di essi aumenterà di conseguenza anche la mole di lavoro da svolgere.

Un altro problema degno di essere sottolineato e causato anch'esso dal passaggio al nuovo linguaggio si è avuto a seguito di alcuni tipi di dato numerici che erano diversi in numero di bit e segno. Nella precedente versione si era usata una maschera apposita per simulare il funzionamento di una variabile intera come variabile binaria cosa che nel nuovo software era possibile effettuare senza espedienti di questo tipo, la scelta è stata quindi quella di rimuovere le vecchie maschere ove presenti e di utilizzare le nuove variabili. Questa scelta ha comportato una ricerca estesa a tutto il progetto ed una revisione di tutte le parti di codice che contenevano tali maschere.
Per molti degli altri bugs riscontrati al momento della prima fase di ricerca a seguito dell'uso della parte d'interfaccia utente molto spesso si è trattato di piccole imprecisioni o sviste e quindi non degne di nota.
Dal punto di vista grafico invece si è dovuto sistemare la parte d'interfaccia allineando i vari componenti e inserendo delle opportune righe di codice per fare in modo che al variare del tipo di centrale ogni elemento presente nelle schede di ogni componente fosse sistemato in modo opportuno.
Durante tutta la fase di debug è stato necessario l'aiuto dell'Ingegner Fusaro e del tutor aziendale per avere informazioni più specifiche nell'ambito del particolare problema da risolvere. Inoltre sono stati necessari dei brevi confronti per chiarire quale fosse la soluzione ottimale da scegliere per eliminare un determinato problema.

4.4 TEST

I test necessari per verificare il corretto funzionamento di una soluzione ad un bug avvengono in gran parte subito dopo che la soluzione stessa è stata implementata. Nel lavoro svolto infatti è stato quasi sempre necessario verificare immediatamente che le modifiche inserite producessero gli effetti sperati in modo da procedere, in caso positivo, con l'eliminazione di altri problemi. Il debug del Programmer infatti è stato un processo di modifica e verifica continuo e progressivo atto a eliminare gradualmente gli errori di codifica o le imprecisioni grafiche che creavano problemi.
Per quanto riguarda i test sulla connessione effettuati una volta che la parte d'interfaccia utente era stata sistemata in tutti i suoi aspetti critici si è dovuto

utilizzare un esemplare di centrale in modo da simulare il collegamento effettivo tra i due elementi: il software in modalità debug e la centrale stessa.
Questa parte non è stata semplice da sistemare perché l'interfaccia scritta in C# che permette la comunicazione tra i due sistemi era regolata da alcune costanti temporali che permettevano al protocollo di effettuare lo scambio di messaggi in maniera opportuna; la verifica "istruzione per istruzione" non era di conseguenza possibile perché i timeout necessari scadevano interrompendo così l'esecuzione del software e molte volte causando anche il crash dello stesso.
Non è stato facile quindi impostare le tempistiche adeguate per far si che la connessione avvenisse in modo esatto e per questa parte del lavoro si è deciso di adottare la tecnica del Brute Force (Forza Bruta), tentativo dopo tentativo si sono raggiunte infine le corrette impostazioni.
Dopo questa prima parte di test sulla connessione si sono portati a termine quelli sul download della configurazione presente in una centrale per poter ricreare l'impianto sul Programmer. In questa fase sono emersi nuovi bugs probabilmente presenti già nella versione precedente del software ma che come già detto in precedenza venivano accettati dal vecchio VB senza creare interruzioni al funzionamento del programma. Per eliminare questi problemi si è dovuto analizzarli caso per caso come per i bugs trovati nella parte d'interfaccia utente. Ad ogni soluzione introdotta corrispondeva una successiva verifica per controllare se la sequenza di download dei dati andasse a buon fine o per lo meno superasse indenne il passaggio appena modificato.
Una volta ottimizzato lo scarico dei dati dalla centrale al software si è passati ad introdurre delle modifiche alla configurazione presente nella centrale utilizzando i dispositivi fisici (tastiere) e non il software, per poi riprovare a effettuare i download e verificare che tutto avvenisse ancora in modo corretto.
In questa parte di prove specifiche tra centrale e Programmer è stato fondamentale l'uso del software XVI32 necessario per controllare che nello scambio di messaggi non vi fossero errori. Inoltre la creazione automatica da parte del software di un file di LOG è stata enormemente d'aiuto. Il file infatti conteneva la registrazione cronologica delle operazioni che vengono eseguite dal protocollo di trasmissione e quindi nello specifico i messaggi scambiati (in figura un esempio).

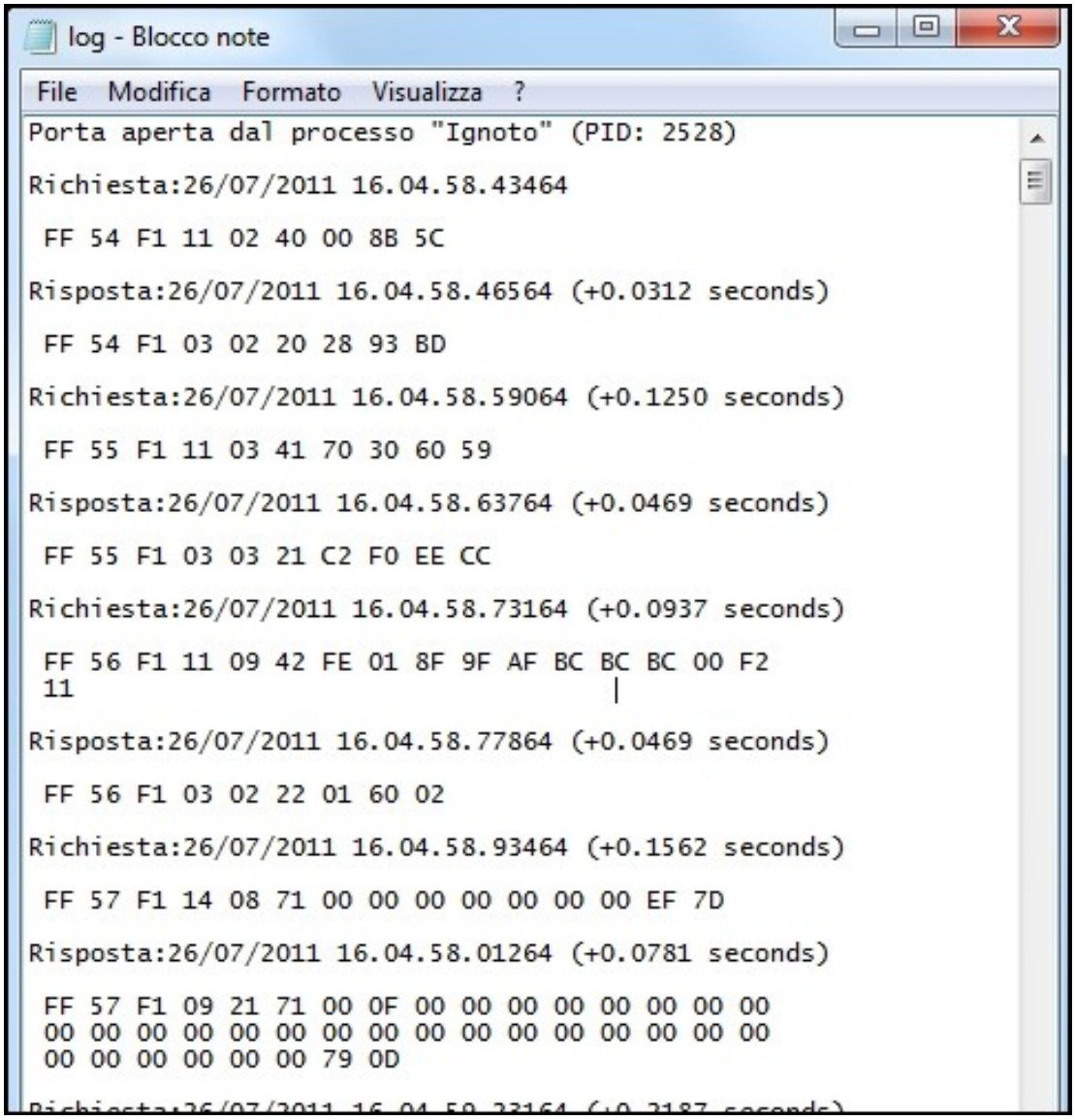

Figura 25: File di Log della comunicazione tra centrale e software

Per ragioni di tempo l'esperienza è terminata con questi test senza testare la funzionalità contraria ovvero l'upload di configurazioni presenti sul software.
Erano necessari inoltre dei test sulle modalità di funzionamento "Monitoraggio Centrale" e "Clonazione di Tastiera".

5 | CONCLUSIONI

5.1 CONSIDERAZIONI SUL PROGETTO

Dopo quanto descritto nel capitolo precedente alla fine del periodo di tirocinio non si possono considerare raggiunti tutti gli obbiettivi del progetto intrapreso dall'azienda, anche se va sottolineato che la ristrutturazione del software è stata portata avanti in modo robusto e che sono stati apportati numerosi miglioramenti. Si sperava che la fine del periodo di stage coincidesse con la finalizzazione del progetto stesso ma, nonostante la pianificazione per fasi del lavoro, il team si è presto reso conto che quanto sperato all'inizio dell'esperienza era stato sottovalutato in termini di tempo. É stato di conseguenza necessario apportare un ridimensionamento delle aspettative nel corso dell'attività di stage.
Per quanto riguarda la parte di interfaccia utente del Programmer Sicurezza sono stati eliminati tutti gli errori derivanti dall'impostazione o dalla modifica di parametri e di quelli dovuti al caricamento e alla visualizzazione di alcuni dati errati in certe particolari configurazioni di impianti.
Dal punto di vista grafico è stato riportato tutto ad una visualizzazione ottimale ed è stata corretta la posizione dei vari elementi nelle forms. Inoltre sono state eliminate alcune imprecisioni grafiche presenti nella vecchia versione oltre ad essere apportati degli accorgimenti e miglioramenti visivi per aiutare l'utente nella fruizione del software e proponendo un utilizzo più intuitivo dello stesso.
Nella parte finale del tirocinio si è portata avanti l'attività di test e debug della connessione e trasmissione di dati tra centrali e pc. Dopo il rallentamento iniziale dovuto ai problemi di connessione già descritti si è potuto procedere con le prove e il debug della parte di software interessata da questa funzionalità. Il download da centrale è stato completamente ripristinato permettendo lo scarico dei dati e la visualizzazione della corretta configurazione della centrale collegata. Mentre per quanto riguarda la correzione della procedura di upload, essa non è stata terminata per sopraggiunti limiti di tempo.

5.2 CONSIDERAZIONI PERSONALI

Nonostante non siano stati raggiunti tutti gli obbiettivi prefissati all'inizio dell'esperienza sono comunque soddisfatto dei risultati ottenuti. Nel corso dello stage ho potuto inserirmi nell'ottica di un'azienda di grosse dimensioni e nel campo dello sviluppo di software, ho potuto ampliare le mie conoscenze e allargare le relazioni con i dipendenti dell'azienda a me più vicini non solo dal punto di vista lavorativo. In questo periodo ho imparato in che modo si affronta un progetto complesso in ambito professionale ed ho potuto utilizzare dei concetti già appresi nel corso di studi come ad esempio la programmazione orientata agli oggetti ed altri paradigmi visti nel corso di Fondamenti di Informatica I, l'uso di protocolli di comunicazione affrontato in quello di Reti di Calcolatori o le tecniche, le fasi e tutti gli altri aspetti legati allo sviluppo di software studiati nel corso di Ingegneria del Software.

Ho avuto inoltre l'occasione, visto che la maggior parte del tirocinio era imperniato su di esso, di approfondire il mondo del debugging. Fino a prima di questa esperienza infatti non avevo avuto modo di conoscere ed utilizzare tutti gli strumenti che servono per affrontare questa problematica e che vengono messi a disposizione per gli sviluppatori dagli IDE di ultima generazione, soprattutto per il fatto di non aver mai dovuto spendere molto tempo in questa attività visto la limitata estensione dei progetti sviluppati in precedenza.

Per quanto riguarda l'ambiente di lavoro nonostante la presenza fissa del tutor aziendale e dell'Ing. Fusaro mi sono accorto che affrontare un progetto di tali dimensioni non è cosa da poco soprattutto se si lavora da soli su di un particolare aspetto di esso, come nel mio caso. Inoltre occorre possedere una buona capacità decisionale, di pianificazione e, non meno importante, di relazionarsi con i componenti del team, utile quando è necessario descrivere le problematiche che spesso sorgono durante la realizzazione del progetto oppure quando vi sono scelte da intraprendere.

5.3 PROBLEMATICHE RISCONTRATE

Come già detto alcune delle competenze richieste ai fini del progetto erano già state acquisite nei corsi presenti nel piano di studi mentre per altre è stato necessario un approfondimento. Rimanendo in questo ambito posso dire che i corsi più propedeutici per affrontare un progetto di questo tipo rimangono quelli di carattere prettamente informatico o tecnologico e che nonostante la presenza di essi la preparazione non era in ogni caso sufficiente ad affrontare le problematiche affrontate. Per questo mi sento di dover dire che troppo spesso nel corso del cammino di studi sia stato messo da parte l'aspetto pratico per privilegiare quello teorico e che una riconsiderazione su questo argomento sarebbe doverosa.

La scarsa conoscenza del software sviluppato e di quello di sviluppo, nonché, come già detto dell'IDE utilizzato nella programmazione hanno fatto si che in molte occasioni abbia dovuto cercare l'aiuto o le delucidazioni del personale coinvolto nel progetto e che questo abbia comportato un costo in termini di tempo da non sottovalutare.

Prendendo in considerazione il software sviluppato esso era affetto da un numero molto elevato di bugs. Questo era dovuto soprattutto dal fatto che era stato elaborato e rimaneggiato da diverse persone in periodi differenti e senza un adeguato scambio di documentazione, dalla fretta di completarlo ed inoltre dal motivo che il software di sviluppo precedente utilizzava delle semantiche di programmazione che ne assicuravano semplicità d'uso, convenienza e rapidità di sviluppo ma che se non venivano usate con i dovuti accorgimenti rendevano possibile lo sviluppo di software apparentemente corretto ma con errori nascosti visibili solamente in fase di utilizzo dalla non ottemperanza alle specifiche richieste.

Se la prima realizzazione del software fosse stata eseguita secondo le regole di programmazione sicuramente nell'aggiornamento previsto da questo progetto si sarebbero incontrate minori difficoltà.

È infatti superfluo ricordare che una buona tecnica di programmazione deve ridurre il tempo passato a ricercare i bugs: programmare rapidamente solo per passare la maggior parte del tempo di sviluppo all'interno del debugger è raramente una tecnica vincente.

5.4 SVILUPPI FUTURI

Dopo aver concluso la parte di test relativi alla connessione di tutte le tipologie di centrale con il nuovo software ed il debug delle problematiche eventualmente riscontrate si dovrà progettare, implementare e sperimentare anche un modulo LAN per la connessione tramite banda larga un'innovazione già da tempo pianificata.
Visto inoltre la veloce esplosione dei dispositivi mobile verificatasi negli ultimi anni tra i potenziali sviluppi futuri del software vi è la possibilità di realizzare un'applicazione web specifica che permetta di accedere al software di controllo utilizzando un qualsiasi dispositivo mobile tramite browser.

BIBLIOGRAFIA E SITOGRAFIA

[1] Manuale d'uso e installazione centrale antintrusione Defender

[2] Manuale Protocollo di comunicazione adottato nelle centrali Defender

[3] VB .NET Developer's Guide

[4] Roger S. Pressman, Principi di Ingegneria del Software, IV edizione, McGraw Hill

[5] http://it.wikipedia.org/

[6] http://www.eptacom.net/pubblicazioni/pub_it/debug.html

[7] http://programmazione.html.it/guide/leggi/42/guida-visual-basic-net/

[8] http://www.deathlord.it/

RINGRAZIAMENTI

Un particolare ringraziamento va ai miei familiari, ai miei genitori Antonio e Silvana, a mia nonna Imelda, a mio fratello Alessandro e alla mia ragazza Eleonora per avermi permesso con il loro sostegno di poter intraprendere e portare a termine questa esperienza universitaria.
Ringrazio il mio relatore, il Professor Marcello Dalpasso, per la disponibilità dimostratami nel seguirmi durante la stesura e correzione di questo documento.
Ringrazio la Fracarro Radioindustrie SpA per avermi dato la possibilità di effettuare questa esperienza ed in particolar modo il mio tutor aziendale, l'Ing. Buson, e l'Ing. Fusaro per avermi accolto e seguito nel corso di esso.
Ringrazio infine i miei amici e tutti i colleghi di studi con i quali ho avuto modo di condividere la mia carriera universitaria, ma in particolare Luca, Fabio, Francesco ed Alessandro con i quali ho collaborato più assiduamente per raggiungere questo obbiettivo.