UNIVERSITA' DEGLI STUDI DI VERONA

FACOLTA DI SCIENZE MM.FF.NN.

Corso di laurea in informatica

Progetto di reti di calcolatori

Utilizzo di Internet per l'automazione e il controllo dei processi industriali basati su tecnologia P-NET

Autori:
Stefano Caldrer
Alessio Fornasiero
Walter Pavani

Novembre 1999

Un caso di studio

Introduzione

L'obbiettivo che ci siamo proposti di raggiungere in questo lavoro consiste nel riuscire a creare un sistema che permetta il controllo di dispositivi fisici (tipicamente utilizzati in ambiti industriali), da una postazione locale oppure remota. Quello che vogliamo fare, consiste nel comandare da un PC un insieme di dispositivi quali motori, sensori, ecc… e successivamente vogliamo rendere tale PC "pilotabile" da un qualsiasi altra macchina che si connetta ad esso tramite Internet.
Dallo studio sulla fattibililà effettuato in prima analisi è emersa la necessità di dividere il problema in due parti

Analisi preliminare

La prima riguardava l'esigenza di trovare un modo per mettere in comunicazione dispositivi fisici di varia natura con uno o più personal computer.La seconda parte consisteva nel riuscire a pilotare il PC connesso ai dispositivi, attraverso un sistema remoto.

La prima parte dello studio ha avuto quindi lo scopo di individuare una rete adatta alle nostre esigenze. Tale rete non poteva essere scelta tra quelle ordinarie che siamo abituati a studiare (Ethernet, token ring etc.) in quando queste consentono la comunicazione esclusivamente tra PC.

Tra le soluzione presenti sul mercato abbiamo scelto la rete P-NET perchè ci è risultata essere la più adatta ai nostri scopi. Le caratteristiche principali di questa rete sono facilità di connettere tra loro dispositivi di tipo diverso (sensori intelligenti, dispositivi di I/O, PC, etc) e relativa semplicità del protocollo di comunicazione

Per il controllo remoto invece il problema consisteva nell'individuare un modo di monitorare ed eventualmente pilotare il sistema in qualsiasi momento e in qualsiasi luogo utilizzando solamente una macchina in grado di connettersi al PC. Per rendere possibile questo si è deciso di utilizzare la rete Internet con il protocollo HTTP, che ci permette di pilotare mediante WEB il PC connesso alla rete P-NET.

Questa soluzione ci è sembrata molto interessante. Attualmente è oggetto di numerosi studi e sta avendo enormi sviluppi: è previsto infatti che a breve termine anche i classici elettrodomestici delle nostre case (frigoriferi, forni, televisori, ecc.) saranno connessi a Internet .

Tra gli strumenti che oggi ci sono a disposizione per scrivere applicazioni di comunicazione via WEB, la scelta finale che abbiamo preso è stata quella di sviluppare l'applicativo in ambiente Microsoft utilizzando PWS (Personal Web Server) e ASP (Active Server Pages).

Specifiche progettuali

Per realizzare quanto detto precedentemente abbiamo implementato, in collaborazione con una ditta operante nel settore dell'automazione industriale, un sistema dimostrativo composto dai seguenti dispositivi:

- PC Master: Macchina sulla quale gira l'applicativo in grado di monitorare e pilotare i dispositivi;
- due Tank Sampler: motori passo passo che servono per prelevare dei campioni di liquido da un condotto;
- una sonda intelligente di temperatura;
- una sonda di livello;

Ogni dispositivo è connesso alla rete e viene pilotato dal PC. Il sistema pone l'utente in grado di selezionare la velocità di rotazione dei motori nei seguenti modi:

- da selettore manuale posto sul tank sampler;
- dal PC attraverso degli appositi scroll-bar;
- da un timer che ne programma il funzionamento;
- da informazioni prese direttamente dall'ambiente (temperatura o livello campionati da sensori);
- attraverso pagine WEB utilizzando form di lettura dati;

Grazie a queste possibilità riusciamo quindi a simulare il sistema di controllo di un processo industriale, controllo che può essere ad opera di un utente o può essere effettuato direttamente dal computer sulla base di alcuni parametri.

Interfacciamento P-NET con iL Personal Computer e i dispositivi

La rete PNET è dotata di un sistema operativo che risulta essere compatibile con macchine DOS o MS-Windows.

Lo schema di funzionamento è stato schematizzato a strati in maniera da risultare analogo alla pila di protocolli del sistema ISO-OSI.

Partendo dagli strati inferiori troviamo l'hardware per la connessione alla rete, nel caso del PC sono disponibili delle schede di rete PCI (vedere) o delle schede da attaccare alla porta parallele. Nel caso in cui si volesse connettere alla rete un dispositivo diverso dal PC l'hardware dovrà essere ovviamente fornito insieme al dispositivo.

Immediatamente sopra troviamo, nel caso del PC, il sistema operativo PNET il quale si occupa di ricevere i messaggi generati dall'applicativo e di trasformarli in pacchetti da inviare sulla rete. Il sistema operativo (VIGO) agisce sulla base di informazioni registrate sul MIB (Management Information Base) il quale è costituito da una tabella contenente la struttura delle variabili dei dispositivi connessi alla rete.

VIGO fornisce inoltre un'interfaccia da utilizzare nelle applicazione scritte al livello successivo. Questa interfaccia è costituita da un oggetto OLE (Object Linked Emmbedded) grazie al quale risulta semplice ed efficiente la gestione delle potenzialità della rete attraverso applicazione scritte con linguaggi di alto livello (Visual Basic, Excel Macro, Visual C++, ecc..)

Ulteriori approfondimenti sul pacchetto VIGO verranno effettuati nei capitoli successivi.

Nei componenti di natura diversa invece (come nel nostro caso Tank Sampler e sonde) hardware e software sono specifici del dispositivo ed inglobati ad esso. E' compito delle case costruttrici quindi progettarli e scrivere le variabili di interesse del componente.

Generalmente assieme al dispositivo viene fornito anche uno schema contenente la struttura delle variabili. Questo serve a fare in modo che esse possano essere inserite nel MIB di VIGO operazione che risulta essere necessaria per pilotare i dispositivi con un PC.

L'hardware del dispositivo, dovrà inoltre essere in grado di pilotare gli ingressi analogici opportuni , come per esempio la corrente da erogare ad un motore per farle compiere un determinato numero di giri per minuto.

Nella figura seguente viene riportata, a titolo di esempio, la rappresentazione dello schema di connessione tra un PC e un Tank Sampler attraverso la rete P-Net

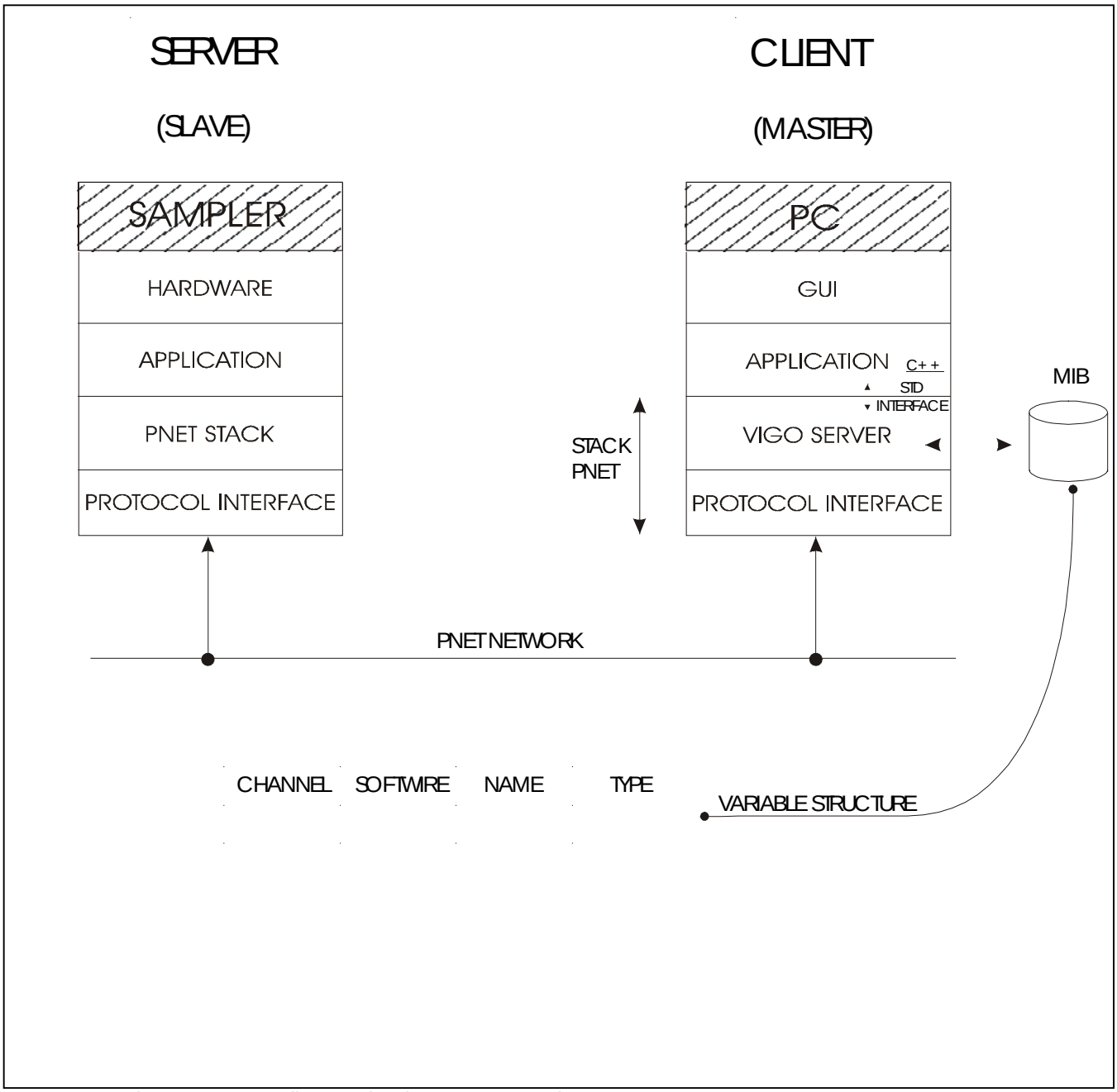


Figura 1 Stuttura di comunicazione PC – Sampler

La rete P-NET

Introduzione

La rete P-NET è una rete concepita per poter connettere insieme componenti analogici e digitali quali sensori intelligenti, moduli di I/O, PLC ecc. Le applicazioni nelle quali l'utilizzo di questa rete è risultato conveniente spaziano dagli ambiti industriali a quelli domestici ad altri di vario genere.

Un primo vantaggio della rete P-NET è costituito dal esiguo numero di fili (due) che sono coinvolti nella connessione dei componenti. I dati, che sono trasmessi in digitale, possono essere di varia natura: valori campionati (da sonde o sensori), informazioni su valori limite, segnali di ritorno, segnali di errore, dati di sistema, dati di configurazione nodi/sensori, caricamento/scaricamento di programmi ecc. .

L'utilizzo da parte di P-NET di sensori intelligenti consente una migliore diagnostica rispetto ai tradizionali circuiti, inoltre il basso numero di collegamenti tra i vari componenti determina un basso costo di cablaggio e di manutenzione.

Gli errori dovuti al cablaggio o al mal funzionamento di dispositivi sono individuati da un appropriato protocollo di rete.

La rete P-NET è adatta indistintamente sia a piccoli che a grandi sistemi.

Una tipica applicazione P-NET gestisce tempi di risposta nell'ordine dei ms, pertanto nei sistemi che richiedono tempi di risposta più rapidi la rete P-NET non risulta essere adeguata.

Caratteristiche tecniche

La rete P-NET è basata sullo standard RS485 , per il cablaggio viene usato un doppino che permette, grazie alle sue caratteristiche, collegamenti lunghi fino 1200 m senza l'uso di ripetitori, contenenti al più 125 dispositivi per segmento.

I dati vengono spediti in modalità asincrona utilizzando la codifica NRZ e possono essere in formato floating point (per valori di temperatura, pressione, corrente ecc.) oppure divisi in blocchi da 32 segnali binari indipendenti che indicano stati della valvola posizione di interuttori ecc. .Le prestazioni partendo da qualsiasi punto del sistema raggiungono i 9.600 segnali binari al secondo.

Questo tasso di trasferimento dati viene raggiunto in quanto i dispositivi slave elaborano i frames, di trasmissione e di ricezione, in parallelo. L'elaborazione da parte dello slave inizia non appena arriva il primo byte di dati, mentre molti circuiti dedicati aspettano l'arrivo di tutto il frame.

Questa filosofia consente allo standard P-NET, (che ha un tasso di trasferimento di 76.800 bit/s), di comparare le sue prestazioni con sistemi che hanno tassi di trasferimento molto più elevati (fino a 500.000 bit/s).

P-NET è una rete multi-master in grado di supportare fino a 32 unità master per segmento. La comunicazione è gestita dal master che spedisce una richiesta e dallo slave che la riceve dando una risposta immediata. Le richieste possono essere di lettura o di scrittura. Il funzionamento di Masters e slaves è schematizzato nella figura sotto.

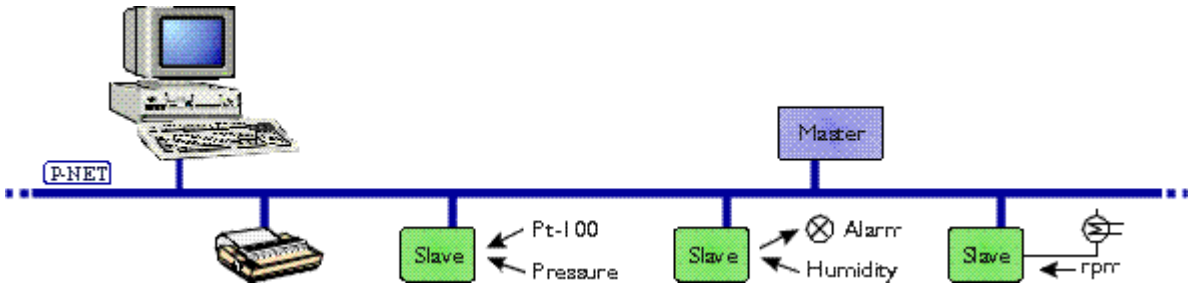


Fig.2 Master e Slave in una rete P-Net

I dati trasferiti possono essere di tipo semplice (boolean, byte, integer, reali ecc.) oppure di tipo complesso (array, string, record ecc.).

Il diritto di accesso al bus viene assegnato ai vari Mater contendente attraverso l'utilizzo di un token. Questo metodo è chiamato "virtual token passing". Quando un master ha terminato l'accesso al bus il token viene automaticamente passato al prossimo master mediante un meccanismo a ciclo basato sul tempo.

Esistono anche altre tipologie di trasporto le quali però richiedono messaggi aggiuntivi per il trasferimento di token. Tutto ciò causa l'aumento del tempo di elaborazione del master e la diminuzione della capacita del bus.

Il protocollo "virtual token passing" gestisce anche il caso in cui il master non sia attivo, in questa situazione tutti i rimanenti dispositivi, compresi gli altri master, funzioneranno normalmente.

Struttura multi-rete

Le strategie utilizzate precedentemente per la costruzione di reti da usare nelle fabbriche erano basate su cablaggi direttamente connessi a sensori ed attuatori.

La fieldbus doveva essere connessa a delle cell-controllers che a sua volta venivano connesse a una cell-network attraverso una gerarchia a livelli. Un livello più alto aveva un'importanza maggiore rispetto ad uno più basso. Questo comportava un'alta velocità sulla dorsale di rete, dove tutti i dati andavano a finire su un potente elaboratore al livello maggiore.

La tecnica di oggi e del futuro è basata su una distribuzione di intelligenza fra cell-controllers, sensori, interfacce. In ogni livello i dati sono concentrati in cicli di regolazione all'interno dello stesso bus.

Non è più necessario avere alti tassi di trasferimento nei livelli maggiori grazie alla distribuzione di intelligenza. Questa è la ragione per cui la rete P-NET è usata in molti settori dell'automazione industriale.

Dividere il sistema in celle indipendenti rende possibile la chiusura di una singola sezione senza influenzare le altre. L'esecuzione di programmi potrebbe essere distribuita in più processi indipendenti.

Gli errori Hardware e software di una cella non avranno effetto sulle altre. Una cella avrà una necessità limitata di scambiare dati con altre celle.

In sistemi con effettiva distribuzione di intelligenza una maggior potenza di elaborazione può essere ottenuta con l'aggiunta di controllori master. Questo implica una possibile espansione del sistema.

Fra i vari sistemi fieldbus, solo P-NET permette un indirizzamento diretto fra vari segmenti di bus che prende il nome di struttura multi-net.

Una struttura multi-net è mostrata nella figura sottostante.

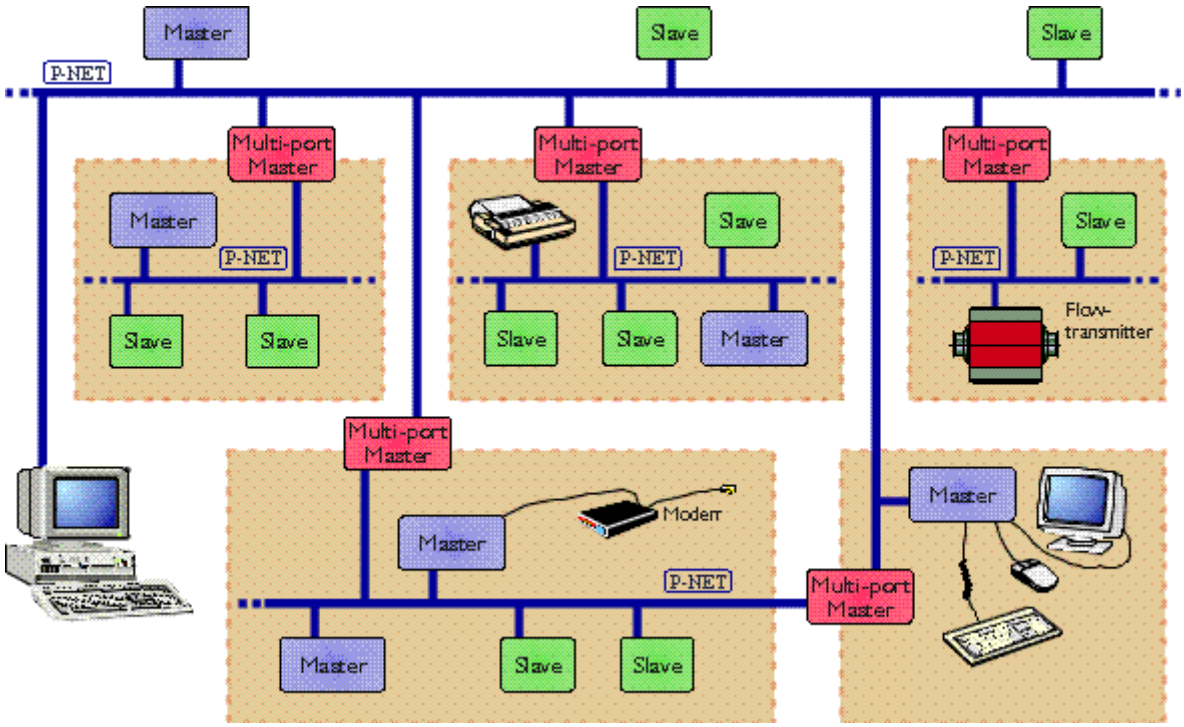


Figura 3 : struttura multi-master con fieldbus P-NET.

La comunicazione è diretta attraverso i vari segmenti di bus per mezzo di nodi (multi-net masters) con due o più interfacce P-NET. Questo significa che ogni master di un segmento può accedere ad ogni nodo ed a qualsiasi altro segmento senza l'utilizzo di programmi speciali. Lo schema di funzionamento è riassunto nella figura sottostante.

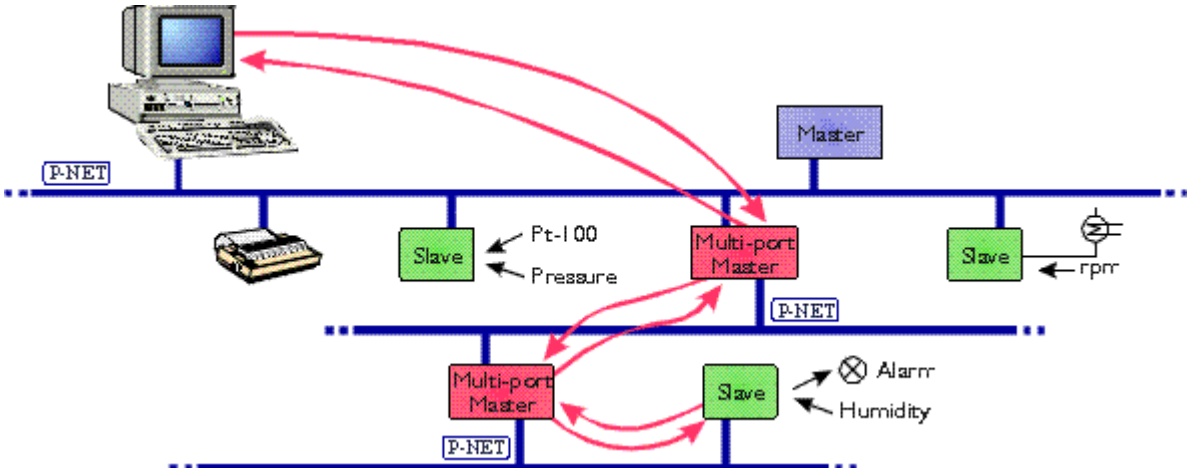


Figura 4 : accesso trasparente mediante master multi-port verso altri bus segment

Un importante vantaggio di multi-net P-NET sta nel fatto che non è strutturata in forma gerarchica, questo permette una migliore e più semplice espansione. Se si vuole connettere due segmenti di bus con un sistema di collegamento che non usa la struttura multi-net è necessario disporre di programmi speciali all'interno del nodo. Tali programmi hanno bisogno di controllare tutti i dati di tutti i dispositivi all'interno del segmento e devono anche scambiare informazioni con altri nodi. Questo determina un consistente uso dei canali di comunicazione con una conseguente diminuzione della capacità del bus.

Questo problema è superato utilizzando P-NET.

VANTAGGI DEL PROTOCOLLO P-NET

Tutti i nodi conformi allo standard P-NET possono essere direttamente connessi al bus e sono in grado comunicare immediatamente, questo perché P-NET usa solamente un data-rate. Questo non avviene negli altri standard che presentano più variazioni in ogni strato con il risultato che non è sempre possibile avere una comunicazione tra tutti.

Nessun modulo P-NET, master compresi, può essere connesso o sconnesso senza interferire con il resto del sistema di trasporto. I moduli possono essere scambiati durante le operazioni di sistema e un sistema può essere espanso finche è in funzione.

La configurazione dei parametri di comunicazione è molto semplice:

- nei moduli slave va solamente settato l'indirizzo del nodo;
- nei moduli master si definisce l'indirizzo del nodo e il numero dei masters.

Quando un dispositivo è connesso con P-NET risultano molto semplici sia le procedure di test fatte durante la fase di sviluppo per programmi all'interno del dispositivo, sia le misurazioni e le procedure di mantenimento che saranno svolte in seguito. Quindi P-NET può essere usata per controllare le variabili all'interno dei dispositivi.

Il risultato di una misura fatto dallo slave è presentata al master in una forma prestabilita. Il beneficio è significativo infatti il master non deve usare scale di conversione aumentando la potenza di elaborazione. Per esempio una misura di temperatura sarà convertita in floating point da parte dello slave e sarà presentata ai masters in gradi centigrado.

Gli identificatori usati per l'accesso alle variabili fisiche nella rete sono mappati attraverso una Softwire List. Questa lista è generata quando il programma di applicazione esegue la compilazione. Quindi non è richiesto un tempo aggiuntivo a vantaggio di un più veloce accesso ai dati.

Ogni frame trasmesso è composto da 56 byte di dati, se il frame è di lunghezza maggiore verrà diviso in blocchi da 56 byte che saranno poi trasmessi.

MODULI P-NET INTELLIGENTI

Un tipico modulo slave P-NET contiene un certo numero di funzioni di I/O.

Esse molto spesso contengono funzioni aggiuntive che variano da semplici controlli su interruttori limite, sistemi per configurare cicli di controllo locali o passi di processo specifici.

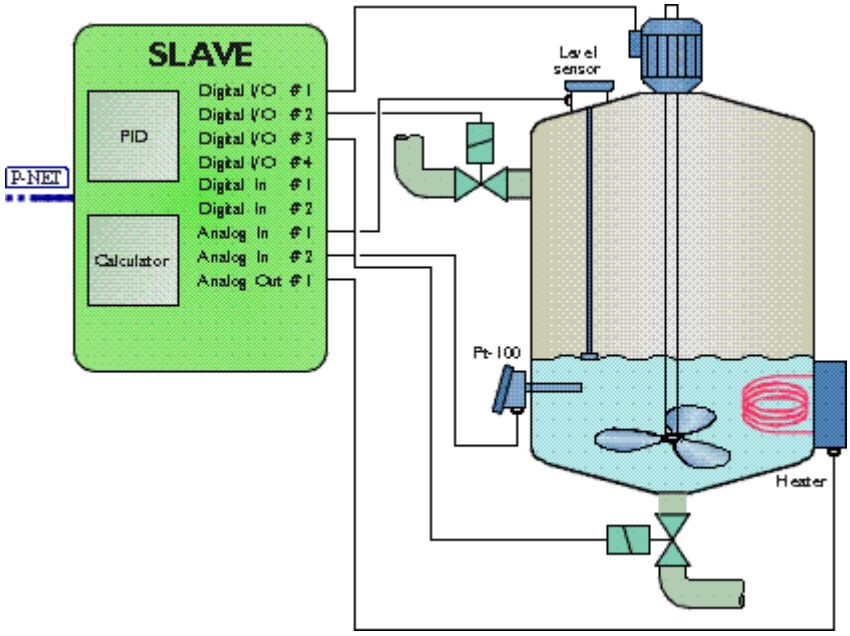


Figura 5: modulo P-NET intelligente di un impianto chimico.

Il diagramma in figura mostra come nello standard P-NET i moduli di I/O possono gestire il controllo della temperatura e il caricamento-scaricamento di liquido per un contenitore riscaldato all'interno di un impianto chimico .

In questo esempio il modulo gestisce la regolazione della temperatura e del livello del liquido e controlla il riempimento. Alcuni valori di temperatura e livelli sono richiesti dal master.

Un altro esempio per il modulo slave può essere un trasmettitore di peso, dove il segnale analogico è in continuazione convertito e memorizzato all'interno della memoria dello slave. Quando viene ricevuta una richiesta da parte del master lo slave risponde con l'ultimo dato memorizzato.

Il controllo degli errori è svolto all'interno dello slave e in caso di errore viene avvertito il master

“Strato 8”: struttura del canale (channel) P-NET

Generalmente i dispositivi utilizzati in P-NET sono sensori, attuatori e interfacce, che sono in relazione con uno o più segnali di processo, cioè un output digitale o un input analogico. Ad ogni process signal sono associate informazioni aggiuntive. Queste variabili sono manipolate da funzioni specifiche per configurazione, conversione, filtro, messaggi di errore ecc. .

In P-NET questa collezione di variabili e funzioni per un singolo process signal è conosciuta come **process object** e è chiamata **channel.**

Un channel contiene tutti i dati necessari per gestire le funzioni di controllo richieste per il process object.

Un channel è costituito da 16 registri ognuno dei quali ha un proprio indirizzo logico chiamato software numbers (SWNo).Queste 16 variabili o costanti all’interno di un channel possono essere di vari tipi e possono essere immagazzinate con differenti tecnologie di memorizzazione.

In figura è mostrato un esempio di interfaccia channel I/O digitale standard .

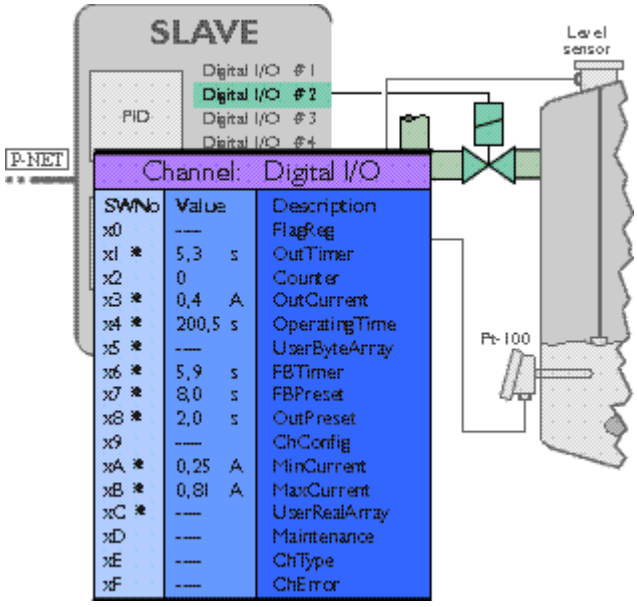


Figura 6: struttura del channel di un I/O digitale.

Un simile channel può essere configurato per svariate funzionalità incluse funzioni automatiche. Queste funzioni sono di input, output, one shot output, timer output ecc.

La funzione è selezionata impostando un codice nel registro ChConfig. Quando l’output è configurato per funzioni timer sono usati i registri SWNo x7 e x8.

Finche il pin di I/O è attivo il registro Operating-time misura il tempo e in presenza di un record di attivazione input o output verrà incrementato il registro Counter.

La corrente elettrica del carico di output è misurata e indicata nel registro SWNo x3.

I registri xA (MinCurrent) e xB (MaxCurrent) possono essere usati come una specie di segnale di ritorno per vedere se il carico è connesso e per altre verifiche di protezione.

Il registro Maintenace può contenere informazioni sull'ultima manutenzione che è stata eseguita per la valvola connessa. Il registro ChTipe deve essere presente in tutti i channels. Esso è un Record, consistente un numero unico, che definisce il tipo di channel più un array di boolean che indicano quanti registri sono implementati.

I registri segnati con " * " non sono obbligatori e possono essere dichiarati non in uso.

Uno degli aspetti più importanti in P-NET è la gestione dei messaggi di errore.

Ogni channel ha un registro chiamato ChError che contiene informazioni su errori relativi al channel e i valori dei suoi registri. Errori tipici sono il sovraccarico, sconnessione segnale ecc.

Un channel è possibile non solo trattare con process signals ma anche con altri generi di dati, come function-blocks interne con parametri corrispondenti.

Un esempio di un simile channel è un PID reguletor dove i valori di output sono il risultato di un calcolo. Altri esempi di channel standardizzati includono channel di stampa, di comunicazione, di programma ecc.

Ciò significa che i nodi possono avere diverse strutture di I/O. Per esempio un nodo può avare 16 I/O channel digitali + 2 analogici, un altro nodo 8 I/O channel digitali + 4 analogici, ma ogni singolo I/O di tipo uguale sarà visto in modo analogo da parte del master, non importa quale specie di nodo appartiene.

Il **Service channel** è un importante channel standardizzato che può essere incluso in tutti i nodi siano essi una collezione complessa di differenti channels o sensori semplici.

Questi channel contengono informazioni su indirizzo del nodo, numero seriale, identificativo costruttore, dati di errore dei nodi complessivi e tutti gli altri dati

associati con i nodi. Questo channel ha sempre un SWNo di 0 e l'accesso al Service channel è lo stesso per tutti i nodi. Questo channel viene usato per identificare un nodo non conosciuto.

Il risultato di questa filosofia di standardizzazione channel è che dal master ogni channel può essere visto e trattato allo stesso modo, non importa chi costruisce il dispositivo e in quale nodo e posizionato.

La standardizzazione rende inoltre possibile scrivere programmi generali che possono essre usati per configurare tipi di channel, leggere o scrivere all'interno dei registri, indipendentemente da dove è posizionato il channel.

Accesso a P-NET tramite PC

I PC sono spesso usati in P-NET come dei masters e sono normalmente connessi attraverso una plug in cards.

Un importante prodotto chiamato VIGO è stato sviluppato per P-NET. VIGO è un sistema di controllo di fieldbus basato su PC . Esso permette ad un impianto fisico di essere descritto in termini di dati, strutture dati connesse e dove i dati sono posizionati.

VIGO è inoltre un sistema di comunicazione che gestisce la sicurezza e l'integrità dei dati all'interno dell'impianto.

VIGO tiene traccia della relazione fra l'oggetto fisico all'interno dell'impianto e i nodi di fieldbus associati. Esso inoltre include una serie di files describing di programmi di controllo connessi, parametri di taratura e configurazione, tool di configurazione, backup, download ecc.

Il routing e il trattamento simultaneo di svariati pacchetti di informazioni, per la stessa o altra rete, sono gestiti da VIGO attraverso una comunicazione in tempo reale. Quando diverse applicazioni tentano di accedere allo stesso bus accadono dei problemi in un ambiente Windows multi-task . Questo è risolto da VIGO che garantisce ai pacchetti e messaggi di comunicazione di non essere mescolati.

Un modo che consente un veloce scambio di dati fra applicazioni MS-Windows è chiamato "OLE2 automation" (Obgect Linking Embedding). VIGO è un OLE2 automation server che genera un'interfaccia consistente e trasparente fra il programma applicativo e l'elemento fisico (object) all'interno dell'impianto.

In questo modo VIGO fornisce una semplice interfaccia a pacchetti standard come Visual Basic e Visual C++, database ecc.

Tutte le operazioni di un object fisico sono fatte attraverso un oggetto virtuale. Vedere figura sotto.

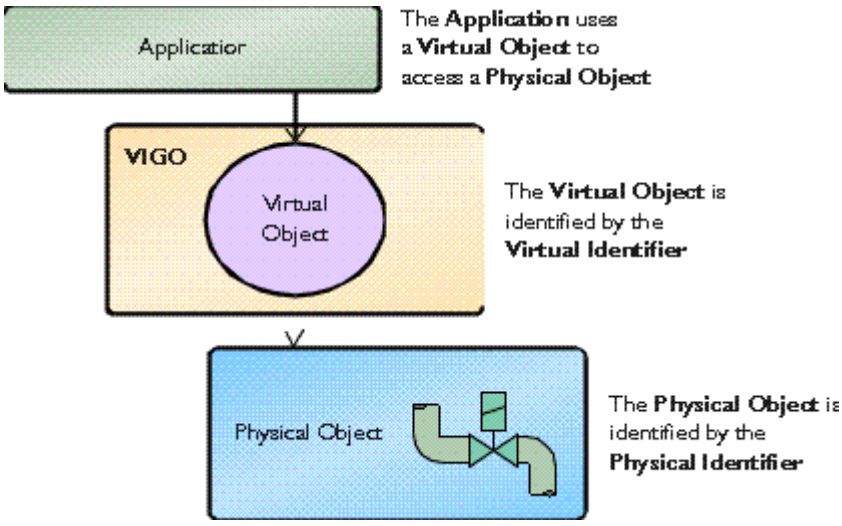


Fig. 7 : Struttura della comunicazione con un PC

Gli oggetti possono essere usati per assegnamenti normali e più oggetti possono essere creati per varie applicazioni indipendenti. VIGO è una collezione di diversi elements program (Applications, VIGOSERV, IDC, HUGO2, The Manager Information Base, The Drivers). VIGO è un sistema aperto che consente l'aggiunta di moduli di rete anche di altri venditori.

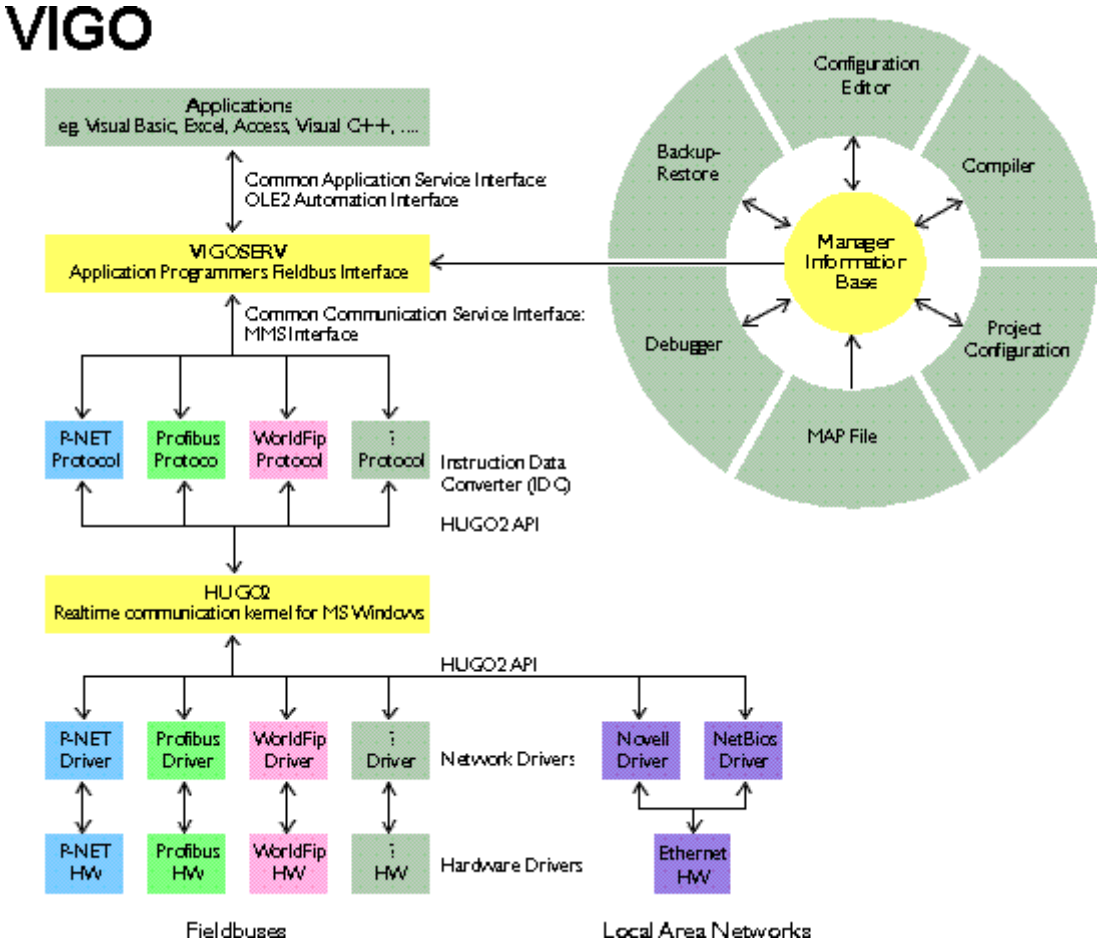


Figura 8 : elementi di VIGO.

Software

Oltre allo standard OLE2 esistono, per P-NET,drivers per DDE (Dynamic Data Exange).

Sono disponibili per P-NET tool software per monitoraggio e debugging, sistemi di controllo grafico, tools per scaricamento programmi e per configurazioni, editor ecc.

Implementazione di P-NET

Una delle ragioni dell'alto numero di installazioni di P-NET risulta il basso costo di implementazione dei nodi.

Il principio di P-NET consiste nell'usare lo stesso microprocessore per controllare il main task dei nodi e i task di comunicazione. Vedere figura sotto a sinistra.

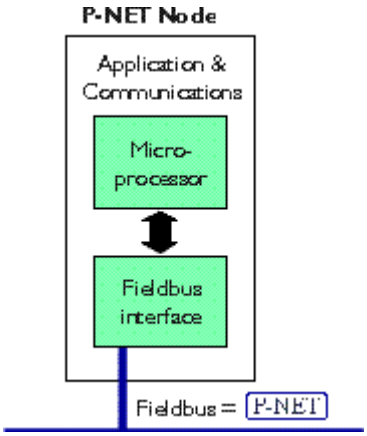


Figura 9 : implementazione di P-NET.

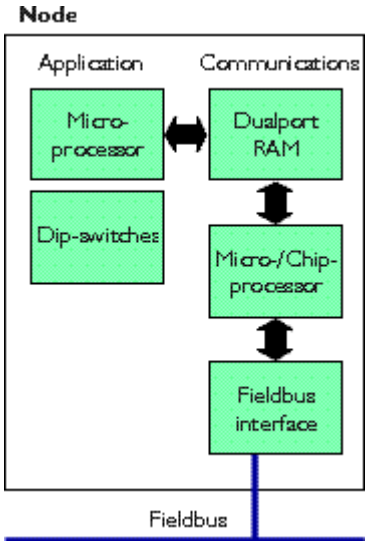


Figura 10: implementazione con chip

Altri tipi di fieldbus usano un circuito addizionale in ogni nodo e questo comporta un alto costo per i produttori finali. Vedere figura sopra a destra.

Con P-NET non c'è bisogno di uno specifico chip/set perché il programma di comunicazione P-NET per uno slave richiede solo un pochi kbyte di codice.

Questo fornisce l'opportunità di usare un microprocessore singolo comune.

Molti anni di esperienza sono stati un guadagno di implementazioni di nodi P-NET e un'assistenza per i produttori è ora disponibile presso International P-NET User Organization.

Architettura P-NET

La rete P-NET è specificata e implementata in accordo con il modello di riferimento OSI (Open System Interconnection) nei strati 1,2,3,4 e 7 come mostrato nel diagramma sotto.

Normalmente un fieldbus è implementato solo negli strati 1,2 e 7 ma poiché l'aspetto tipico di P-NET è la struttura multi-net, il protocollo implementa anche gli strati 3 e4.

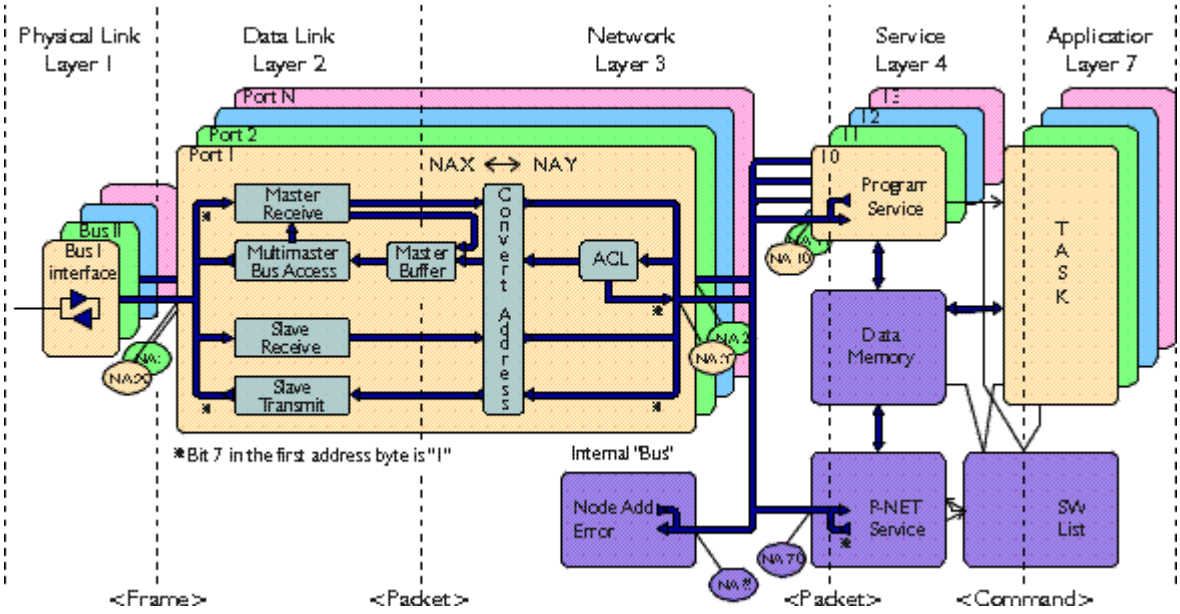


Figura 11: architettura P-NET basata sul modello di riferimento ISO.

Strato 1 : è interessato alla trasmissione di bit sul bus. Specifica il cablaggio, come viene rappresentato un "1" e uno "0", qual è il livello di voltaggio ecc.

Strato 2 : si occupa del multi-master token, impacchetta i dati per essere spediti all'interno del frame, indirizzo sorgente e destinazione inclusi ed esegue l'error detection.

Strato 3 : è il P-NET "post office", il quale si occupa di spedire e ricevere frame in accordo con l'indirizzo di destinazione. Un messaggio può essere richiesto per essere spedito fuori da un'altra porta P-NET, o dentro il P-NET service, o ritornare all'applicazione richiesta, o ritornare un messaggio di indirizzo non conosciuto. E' inoltre eseguita la conversione di indirizzo necessaria per garantire una risposta.

Strato 4 : tratta due differenti aspetti. Il primo riguarda il P-NET service, il quale legge o scrive dati nella memoria interna attraverso una lista SOFTWIRE, o devia una richiesta se la lista SOFTWIRE indica che le variabili sono posizionate in altri nodi. Il secondo tiene traccia circa il numero di richieste che sono state spedite ma che sono il attesa di una replica. Quando la replica arriva esso spedisce una calling application task di ritorno.

Strato 7 : è usato dai programmi applicativi per accedere a variabili in altri nodi. Questo è svolto spedendo un command block contenente riferimenti alla SOFTWIRE list la quale contiene informazioni dettagliate su indirizzo dei nodi, indirizzo interno ecc. La SOFTWIRE list è inoltre usata per variabili interne.

Passaggio di token virtuale

A ogni nodo P-NET è dato un indirizzo (NA) fra 1 e il numero di master all'interno del sistema.

Tutti i master contengono un "idle bus bit period counter" (contatore di bus inattivo) che incrementa per ogni bit period di bus inattivo, ma è resettato a zero quando il bus diventa attivo. Ogni master ha anche un contatore di accesso il quale è incrementato quando il "idle bus bit period counter" raggiunge 40, 50 ,60,….

Quando il contatore di accessi in un master è uguale all'indirizzo del nodo, il master prende il token e gli è permesso l'accesso al bus. Quando il contatore di accessi supera il numero di master viene riprogrammato a 1.

Il diagramma sotto mostra un esempio del principio del token in P-NET all'interno di un sistema con 4 master.

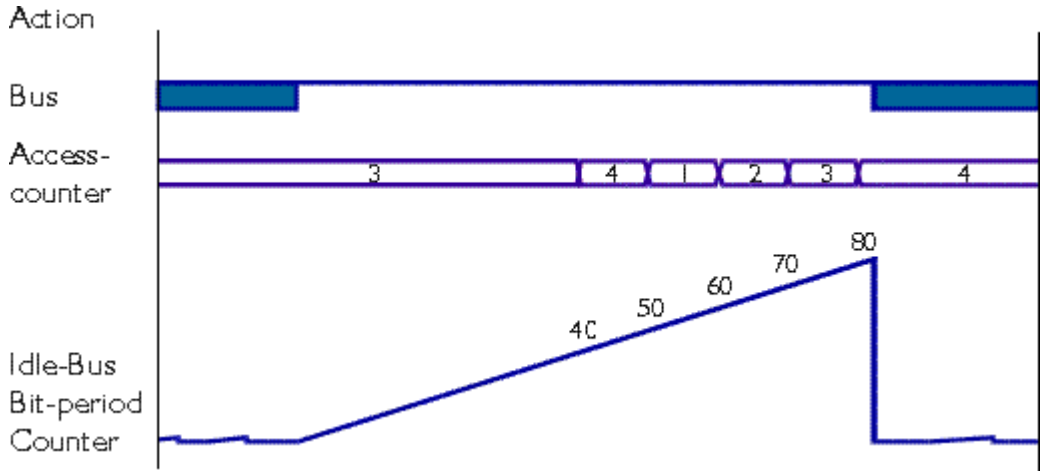


Figura 12: passaggio del token virtuale.

All'inizio il master 3 possiede il token e riceve una risposta dallo slave. Allora il bus diventa inattivo. Quando sono stati contati 40 bit periods inattivi tutti i contatori di accesso sono incrementati di 1 e al master 4 è concesso l'accesso al bus. Poiché il master 4 non ha spedito nulla e sono passati in totale 50 bit periods l'accesso al bus è concesso al master 1. Il master 1 non necessita dell'uso del bus (ricordiamo che il master potrebbe non essere sempre presente) cosi il token virtuale è passato al master 2 quando il bit periods è arrivato a 60.

Poiché i master 2 e 3 non richiedono l'accesso il token è passato al master 4 quando il bit periods è uguale ad 80. In questo momento il master 4 richiede l'acceso, i dati viaggiano sul bus cosi tutti i "idle bus bit period counter" sono portati a zero.

Il passaggio del token virtuale viene eseguito in soli 130 µs o in 10 bit periods e quando non sono presenti dati sul bus. Una singola rete può avere fino a 32 master con la stessa priorità senza bisogno di una gerarchia per essere gestiti.

Di conseguenza P-NET non richiede speciali funzioni per accesso al bus, il passaggio del token virtuale è più che efficiente.

Comparazione fra P-NET e Fieldbus chip dedicati

I sistemi che usano speciali chip ricevono tipicamente il primo frame completo, a questo punto il chip spedisce un acknowledge al master e un interrupt alla CPU. A questo punto partirà l'elaborazione dello slave e quando tutti i dati saranno pronti esso sarà trasferito al chip. Ora il master deve fare una seconda richiesta per ottenere il risultato desiderato. Tutto questo è illustrato nella figura sotto.

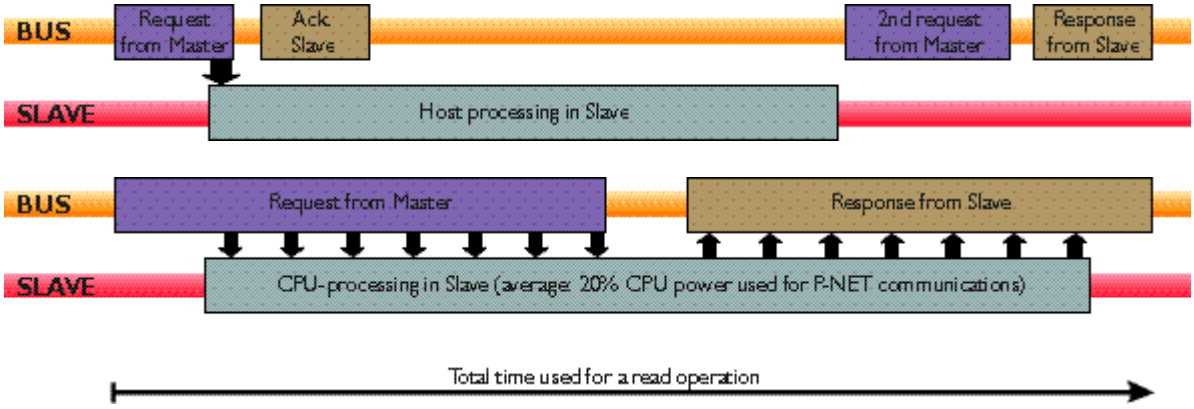


Figura 13: comparazione fra P-NET e fieldbus basato su chip.

Lo slave P-NET tratta l'elaborazione di dati e la trasmissione e ricezione di frame in parallelo. L'elaborazione della richiesta inizia all'interno dello slave non appena arriva il primo byte di dati. In questo modo lo standard P-NET data rate è di 78.600 bit/s non è un fattore limitante in prestazioni.

Il modulo slave P-NET deve sempre rispondere ad una richiesta entro 390 µs. Questo elimina la necessità per richieste multiple per una variabile singola o continui polling non prima che il risultato sia pronto. La risposta immediata elimina la necessità di buffer nello slave per contenere code di richieste o polling da master differenti.

La risposta immediata unita con l'elaborazione in parallelo e il veloce passaggio del token, danno come risultato prestazioni simili ad altri sistemi a bus con un più alto data rate (es. 500 kbit/s).

Uno dei svantaggi dell'incremento del data rate è che esso comporta significante riduzione nella lunghezza del cablaggio del fieldbus permessa. Per esempio con 76,8 kbit/s la lunghezza del bus può essere entro una regione di 1,2 km, ma con 500 kbit/s deve essere ridotta fino a 200 m.

La conseguenza è che per un sistema con prestazioni elevate si devono usare dei ripetitori.

IL PROTOCOLLO

P-NET è una rete concepita per utilizzi in ambiti industriali; riportiamo qui di seguito le caratteristiche principali e il protocollo di funzionamento.

Componenti:

Slaves: Sono i dispositivi connessi alla rete il cui compito è quello di rispondere alle richieste avanzate dai Master.Ogni slave dispone di una mappatura delle variabili globali chiamata SoftWire list. Il tempo medio che impiega uno slave a rispondere è di 150μs mentre il ritardo massimo consentito è di 390 μs a 76,8Kbit/s

Master: Il Master è l'unico dispositivo in grado di spedire una richiesta nel bus, e tale richiesta può solamente essere indirizzata ad uno slave. Il numero dei master normalmente e basso rispetto al numero degli slaves e questi ultimi dispongono di un hardware molto meno performante dei Master. L'idea è infatti quella di far girare sui master i programmi più pesanti e di lasciare agli slave la parte più leggera.

Gateways: La funzione dei gateways e quella di isolare 2 o più bus P-NETe fare il routing dei frame tra una rete e le altre. Un gateway è da una parte slave e master d'altra.

Architettura

L'architettura della rete P-NETè basata sullo standard ISO-OSI e rappresentata nella figura sottostante:

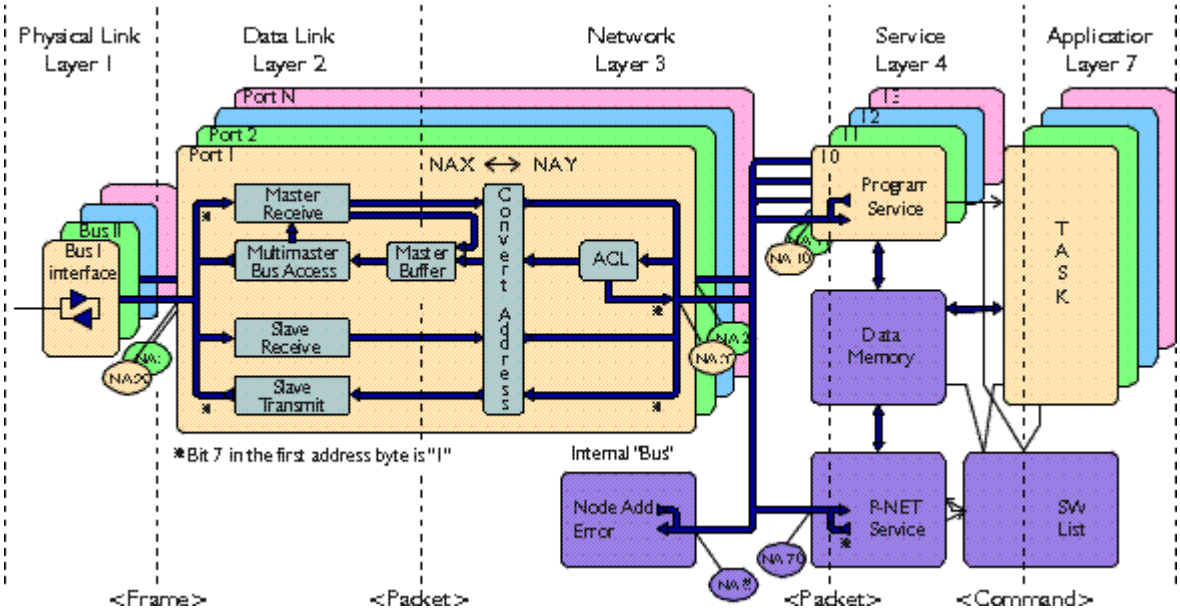


Fig.14: Architettura P-Net stratificata

LAYER1: Livello fisico

Il primo strato dello stack iso-osi descrive come trasmettere le sequenze di bit nella rete, sono specificati: interfaccia elettrica, cablaggio, baudrate e cosi via.

Cablaggio:	Utilizza la standard RS485
Trasmissione:	La codifica utilizzata e la NZR, il segnale non cambia durante un bit-time

Specifiche elettriche:

Struttura del Bus:	Anello fisico senza terminazione
Bus lenght:	max 1200M
Nuero max di device:	125
BitRate:	76.8 kbit/s +/-0.2%

LAYER2: DataLink

I servizi forniti da questo livello sono:

1. Controllare l'accesso al bus
2. Creare e riconoscere il formato dei frames e l'indirizzo
3. Effetuare il controllo di errore

Controllo d'accesso al bus

Tutta la comunicazione è effettuata spedendo frames nel Bus. Il frame viene costruito aggiungendo alle informazioni di base da spedire altri dati aggiuntivi che garantiscono il corretto instradamento del pacchetto

Un frame contiene i seguenti campi:

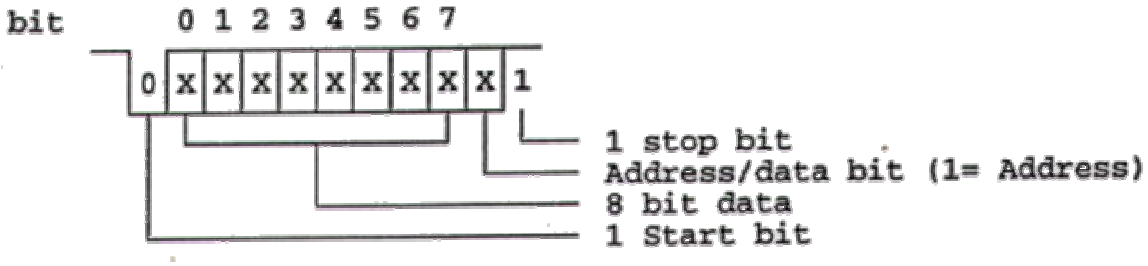
- indirizzo destinatario
- control status
- lunghezza Info
- campo info
- error detection

Node addres field	Control/Status	Info length	Info field	Error detection
2-24 bytes	1 byte	1 byte	0-63 bytes	1-2 bytes

Formato del frame

Tutti i byte del frame sono inviati tramite trasmissione asincrona quindi necessitano di un bit di start ed uno di stop per la sincronizzazione.

Ogni byte inoltre è corredato di un bit (Addres/data bit) che indica ai dispositivi se il byte letto costituisce un indirizzo o meno.



tutti i frames sono separati da un periodo di inattività di 11 cicli di clock.

Error detection

Lo standard P-NETsupprta due differenti strategie per l'error detection

1: Normal error detection: in grado di individuare errori con una distanza di Hamming pari a due, Su una parola di 64bit viene individuato l'errore se questo coinvolge al più 16 bit.

2:Reduced error detection: Il controllo e più leggero ed e in grado di trovare l'errore solo se esso coinvolge al più 8 bit (su una parola di 64)

Ricezione di un frame:

Ciascun nodo e dotato di un unico indirizzo (Node Addres) il cui range varia da 1 a 125 .

I bit 0..6 del primo byte di un frame contengono l'indirizzo del nodo, il bit 7 indica se il frame contiene una richiesta avanzata dal master oppure una risposta dello slave.

Se un nodo riceve un indirizzo uguale a NA o 126 o 127 o NA+128 l'intero frame verrà letto dal nodo. Gli indirizzi 126,127,NA+128 vengono riservati per broadcastiong, testing e routine di gestione interna.

Accesso al Bus

Multiaccesso al Bus da parte dei Master:

P-NET è un sistema multi-master dove a più master è permesso di usare lo stesso bus ma non allo stesso tempo.

Ad ogni master viene assegnato un node-address (indirizzo del nodo), il quale varierà da 1 al massimo numero di master collegati al bus. Il numero massimo non deve superare 32. Il master al quale è permesso utilizzare il bus viene chiamato "Token Master". Il master prenderà così il "Time based Token" in maniera ciclica. Questo significherà che dopo il master 1, il token passerà al master 2, 3,4 e così via. Ad ogni master è permesso spedire solo una richiesta ogni volta che è in possesso di un token (cyclic priority). Ogni master necessita di conoscere il numero massimo di master, perché dopo il master con NA uguale al numero massimo di master collegati al bus il token passerà al master 1.

L’elemento base del multi-master access control è "idle-bus-bit-period-counter" e "access-counter".

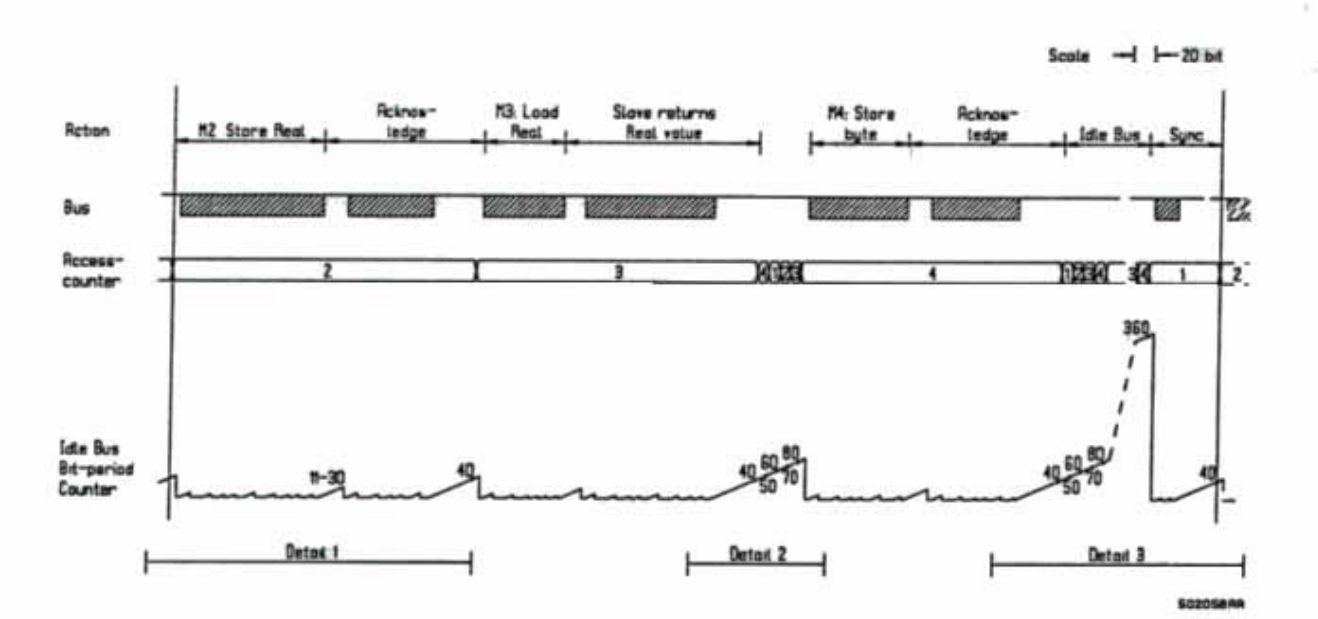


Figure 7: A typical event on a P-NET bus.

Il "idle-bus-bit-period-counter" conta il numero di "1"ni sul Bus e viene azzerato ogni volta che compare uno "0".

L'access counter aumenta quando l’ "idle-bus-bit-period-counter" è uguale a 40-50-60... ed è posto ad 1 quando si è arrivati al massimo numero di master. Vedi fig.7

Quando l'access counter diventa uguale al node address di un master, a questo master è permesso di usare il bus (preso il token) nell'intervallo tra 2 e 7 bit periods dopo il match. Vedi fig.8. Un master solo accede al bus se esso ha una richiesta da spedire.

Sincronizzazione dell' "access counter":

Il primo nodo di ogni frame con il bit 7 = 1 contiene l'indirizzo del nodo del token master. Questo indirizzo viene usato per sincronizzare gli "access counter" di tutti i master.

La perdita di sincronizzazione potrebbe causare trasmissioni sbagliate o crolli della rete.

Slave bus acces:

Ad uno slave è consentito di accedere al bus solamente in seguito ad una richiesta avanzata da un master e il tempo che ha per farlo è compreso nell'intervallo tra l'11o e 30o bit period successivo all'ultimo byte della richiesta del frame.

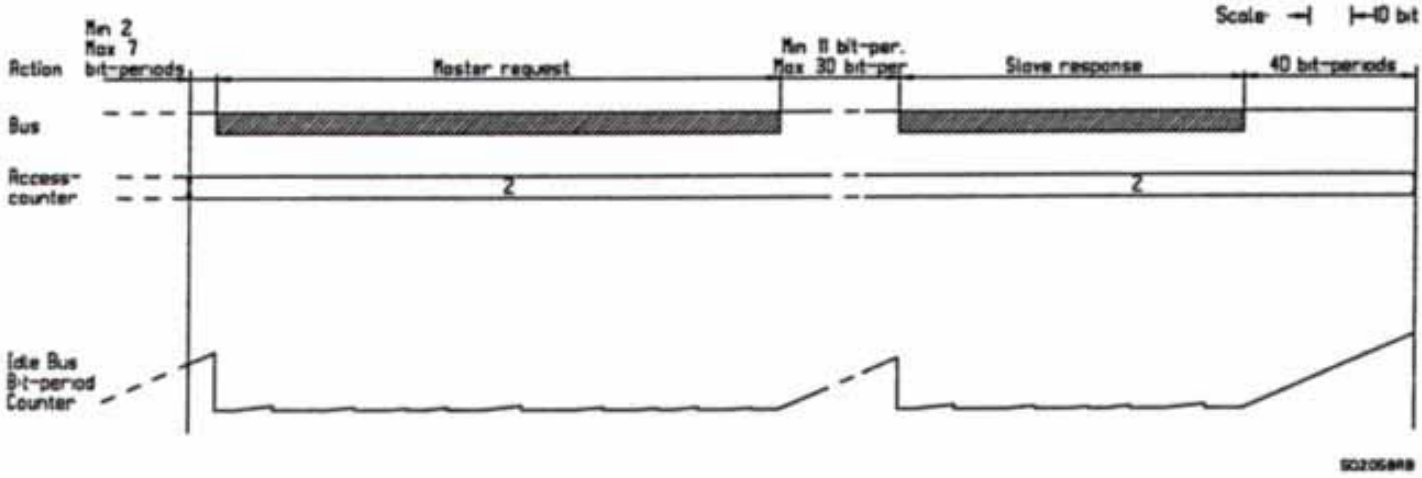


Figure 10: Detail from Figure 7, Slave bus access

Addres field types

Ogni frame della rete P-NETinizia con il campo addres all'interno del quale viene codificata anche la sua lunghezza (variabile da 2 a 24 bytes).

In questo modo si evita di dover spedire un apposito bytes che codifica la lunghezza e che comporterebbe troppa ridondanza nel caso di indirizzamenti corti.

Questo metodo è praticabile solo se si utilizzano un numero di indirizzamenti finiti che nel nostro caso è di 4.

Il Bit 7 dell' addres bytes viene utilizzato per indicare il tipo di indirizzamento e separa l'indirizzo sorgente da quello del destinatario.

I 4 tipi di indirizzamento sono :

1 Simple addres type (Utilizza 2 bytes per source e destination)

2 Complex addres type

3 Extended addres type

4 Response addres type

LAYER 3: the network Layer

Lo scopo del livello di rete è quello di trasportare i pacchetti tra il secondo livello e il 4o. Ciò include Node Addres conversion e la generazione di "Answer Comes Later" . Viene effettuata anche la gestione degli errori e del master buffer.

LAYER 4: The service level

Il compito principale del livello di sevizio sono : utilizzo di una tabella che contiene informazione strutturate sulle variabili globali come indirizzi dei nodi e tipi delle variabili. Queste variabili globali possono essere interne o situata in altri slaves. Questa tabella prende il nome di Softwire-List.

Queste variabili nella sofwire list sono rappresentate attraverso un indice chiamato softwire number (SWNo)

Packet Format

Tutte le informazione nel livello fisico sono contenute in un frame. Il formato di un frame è stato descritto precedentemente. Il livello data link converte dei frame in pacchetti e il formato di quest'ultimo a molto simile al primo.

Il campo node-addres e abbastanza differente mentre l'error detect code viene omesso. La dimensione del campo indirizzo e info field e fissata e infine il pacchetto contiene il retry timer.

Formato:

Node Addres field	25Bytes
Control/Status	1 Byte indica il tipo di operazione (inserire figura)
Info Lenght	1 Byte
Info field	63 Bytes
Retry timer	4 Bytes

LAYER 7: Application Layer

Il livello di applicazione può accedere alle variabili definite nelle softwire list. Queste variabili possono essere locate internamente o in altri nodi. Quando un applicativo sta girando, esso può accedere a tutte le variabili attraverso la SW list.Le Softwire list sono raggruppate in gruppi di 16 che vengono chiamati CHANNEL, questo ulteriore livello di gerachia garantisce un'interfaccia con le applicazioni semplice ed efficiente.

VIGO 4.0

VIGO in generale

VIGO è un sistema di gestione per la rete P-NET. Viene installato su PC che utilizzano Windows come sistema operativo. VIGO è usato insieme a processi di automazione industriale dove le singole unità di controllo sono distribuite su tutto l'impianto e dove uno o più bus sono usati per intercomunicare i dati. Microsoft Windows è un sistema operativo il quale esegue programmi, controlla la tastiera e lo schermo, gestisce il disco fisso e contiene strumenti per configurare ed eseguire programmi. Analogamente, VIGO è un "sistema operativo" usato per maneggiare differenti funzionalità specifiche di un sistema P-NET.

Alcune di queste funzionalità sono:

- Provvedere un collegamento uniforme e ben definito tra i programmi standard del PC, le costanti e le variabili dei nodi presenti sulla rete. Queste variabili e costanti sono identificate da un unico nome (identificatore). Un programma standard potrebbe essere un foglio i calcolo Excel oppure potrebbe essere creato da usando Visual Basic, C++, Delphi ecc.
- Tenere informazioni riguardo la locazione ed il tipo di ogni identificatore. Queste informazioni includono l'indirizzo del nodo per l'interfaccia del modulo, un indirizzo simbolico, l'offset, la struttura dati, il tipo di dati ecc.
- Eseguire simultaneamente comunicazioni tra interfacce di diversi Fieldbus e gestire i problemi di code che occorrono in ambienti multi-tasking, quando diverse applicazioni desiderano comunicare nello stesso momento.
- Tenere traccia degli strumenti che possono essere usati con i vari tipi di dati e le varie strutture dati, considerando l'oggetto e l'interfaccia che si sta considerando. Questi strumenti possono essere compilatori, strumenti di configurazione, assembler ecc.
- Fornire informazioni a compilatori e assembler riguardo le variabili che già esistono in VIGO, che non devono essere dichiarate di nuovo. E' perciò possibile creare compilatori dove non è necessario dichiarare tutte le variabili globali, perché il compilatore stesso può caricare le informazioni necessarie direttamente dalla descrizione che VIGO tiene sull'impianto.
- Fornire un editor ordinato di componenti che mantengono una descrizione del'impianto fisico, dove nodi, tipi di dati e gli identificatori associati sono definiti. Così sarà possibile inserire, modificare o cancellare singoli elementi dalla descrizione usando un altro programma (oltre a questo metodo, sarà possibile anche con gli OLE automation interface).
- Simulare i dati dell'impianto con il PC. Questa può essere realizzata facilmente connettendosi off-line con una configurazione di backup e ripristino delle informazioni dell'impianto.

Ogni scambio di dati tra il programma utente e VIGO è realizzato seguendo le OLE automation (lo standard Microsoft per lo scambio di dati). Come un OLE automation server, VIGO provvede ad aprire e definire interfacce per il programma dell'utente finale. Ogni richiesta di dati da ogni punto della rete dell'impianto sarà trattata e vista come direttamente accessibile dal PC. L'utilizzatore dovrà solamente considerare le variazioni nei differenti protocolli di comunicazione, conversione dei dati e metodo di indirizzamento (cioè considerare dove è collocata la variabile o la costante in uso e quindi a quale oggetto fa riferimento).

Dal punto di vista dell'utilizzatore, ognuna di queste funzionalità può essere maneggiata da VIGO, ottenendo così un semplice, uniforme e ben definita interfaccia di tutti i dati della rete. VIGO da questo punto di vista è considerato un sistema aperto perché possono essere aggiunte senza problemi, nuovi strumenti per la gestione e nuovi fieldbus dall'utilizzatore stesso.

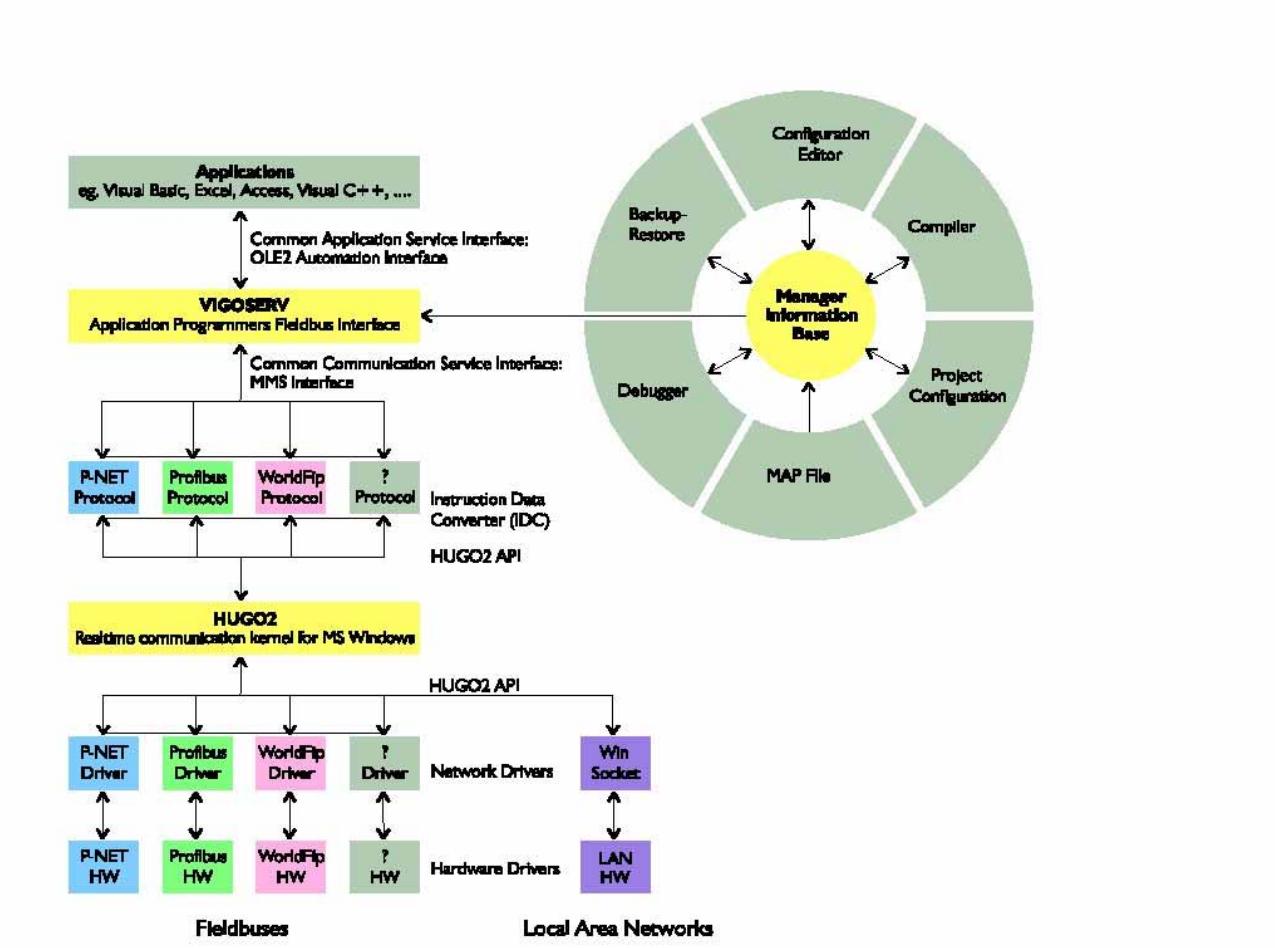
Elementi di VIGO

Il sistema di gestione dei Fieldbus VIGO è una collezione di alcuni programmi. Gli elementi principali all'interno di VIGO sono i *VIGOSERV*, il *MIB* e *HUGO2.* La struttura flessibile di VIGO fa si che aggiungere nuovi elementi sia estremamente facile e cresce con le necessita dell'utilizzatore. Questi elementi, che possono essere collegati dinamicamente senza richiedere cambiamenti al sistema esistente, sono *Convertitori Istruzioni e Dati* (*Instruction Data Converters*), *Driver di Rete (Network Drivers)* e *Driver Hardware (Hardware Drivers).*

VIGO permette all'applicazione dell'utente di essere progettata non dovendo considerare quella che sarà la disposizione della rete, rappresentando quindi la rete come una collezione di componenti indipendenti e installabili.

VIGO permette quindi di aggiungere dinamicamente nuovi strumenti utilizzando il *Node Configuration Editor*, un convertitore di file *MAP*, utility di Backup e ripristino, un Monitor, un compilatore ecc.

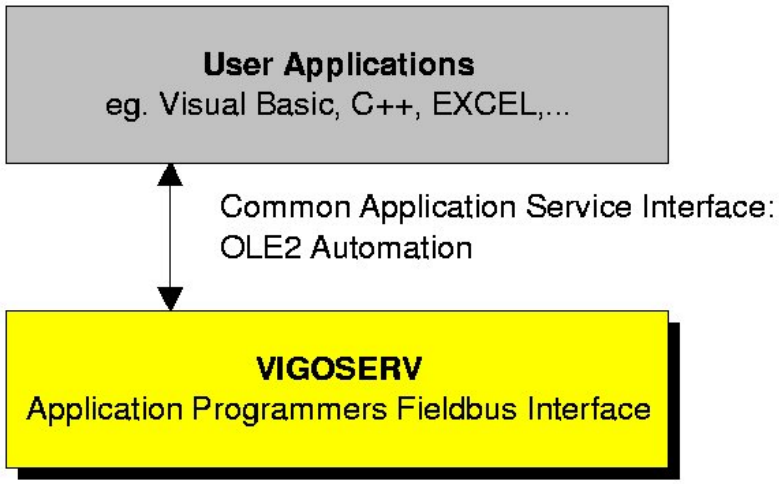
VIGO



Per come è stato costruito VIGO è un sistema aperto, il quale può essere espanso connettendo nuove reti, nuovi oggetti fisici ad una rete già esistente oppure nuovi strumenti per la configurazione. E' considerato aperto perché chiunque può collegare una rete, oppure implementare strumenti per la gestione e tutti possono sviluppare un applicazione che usi le funzioni di comunicazione fornite da VIGO.

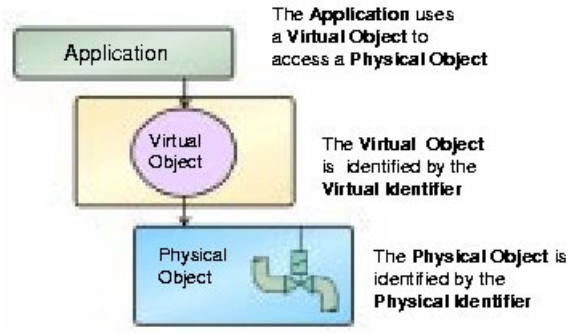
Interfaccia dei programmi applicativi del Fieldbus

L'interfaccia dei programmi applicativi del Fieldbus – VIGOSERV, fornisce una semplice interfaccia a programmi quali ad esempio Visual Basic, Delphi e Visual C++, fogli di calocolo, database e altri programmi di visualizzazione come SCADA. VIGOSERV è un OLE Automation Server, il quale crea una consistente e trasparente interfaccia tra il programma utilizzante (applicazione) e l'elemento fisico (oggetto) dell'impianto.



L'OLE Automation è una parte di Object Linking and Embedding (OLE2), la quale è una funzionalità fornita da Microsoft Windows, per abilitare in tempo reale i cambiamenti di dati tra applicazioni.

VIGOSERV supporta funzioni, come scrittura e lettura di variabili, upload e download di file, inizio,fine e reset di programmi ecc., senza essere a conoscenza di quelle che sono le operazioni della rete stessa. Queste funzioni insieme a tutti i loro parametri, definiscono il Common Application Service Interface. La figura illustra il collegamento tra VIGOSERV e le applicazioni.



La manipolazione di ogni oggetto fisico è archiviata associandola all'oggetto virtuale all'interno di VIGOSERV. Gli oggetti virtuali sono creati dall'applicativo, dove è viene definito un identificatore virtuale (virtual identifier). L'oggetto virtuale è creato indicando l'oggetto fisico per mezzo di identificatori fisici (physical identifier) che dovrà avere un unico nome.
L'identificatore fisico è definito nel MIB.

Manager Information Base (MIB)

VIGO include un Manager Information Base – MIB. Utilizza il MIB per descrivere tutto il sistema di controllo del FIELDBUS , il quale in VIGO viene indicato con il nome di Progetto (Project).

In termini generali, un sistema a Fieldbus è costruito con un insieme di dispositivi fieldbus, chiamati Nodi (Nodes). Il MIB contiene una descrizione dei differenti nodi presenti nel sistema, inoltre mantiene informazioni anche degli stessi nodi come ad esempio l'indirizzo del nodo (Node Addresss), Rete (Nets), l'identificatore del nodo (Node Identifier), il tipo del nodo (Node Type) e altre informazioni rilevanti. Esso mantiene anche informazioni riguardanti la rete all'interno del progetto. Utilizzando tutte queste informazioni sarà quindi possibile calcolare il percorso di comunicazione.

Inoltre un nodo consiste anche di un numero di variabili. Il MIB contiene una descrizione di tutte le variabili all'interno del nodo al quale si può accedere attraverso il FieldBus. Ogni variabile all'interno del nodo può essere di tipo semplice (byte, integer, boolean) oppure di tipo complesso (array, record, string).
In VIGO è possibile vedere le stesse variabili di un nodo come tutte appartenenti ad un tipo complesso, il tipo *Node*. L'acceso viene definito allo stesso modo con cui avviene l'accesso ad un record in Pascal o in C.
Allo stesso modo, tutti i nodi di un impianto possono essere visti come un enorme variabile complessa identificata dal nome del progetto (Project) stesso. Quindi l'accesso al nodo viene visto come l'accesso ad un record dove il Project è il record mentre il Node è un campo all'interno del Record Project.

Un identificatore sarà quindi composto dall'unione i tutti questi costrutti. Si comincerà con l'identificatore del progetto (*Project_Identifier*) segiuto da " **:** ".
In seguito si specificherà l'identificatore del Nodo (*Node_Identifiere*) poi la variabile (*Variable_Identifier*) separate da " **.** ".

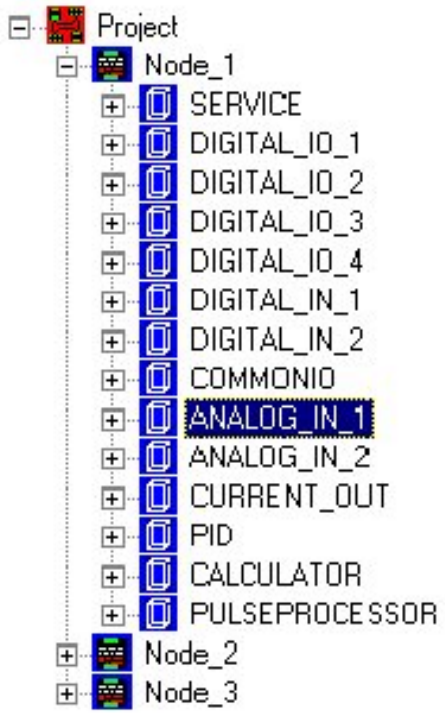
Project_Identifier:Node_Identifier.Variable_Identifier

Così la definizione della Variabile consiste di un Identificatore di Varibile, di informazioni riguardo la locazione delle variabili ed una descrizione del tipo stesso. Tale definizione deve essere disponibile per ogni tipo di variabile, sia essa una variabile semplice, complessa, un Nodo o un Progetto.
Un esempio può essere dato dalla definizione di una variabile. La variabile è identificata da un nome detto *Variable_Identifier*. La locazione della variabile descrive l'indirizzo interno dentro un Nodo. La descrizione di un tipo per una semplice variabile consisterà quindi nella definizione i un tipo di base. Stessa cosa per i Nodi. L'indirizzo questa volta descriverà il percorso del fieldbus dei nodi. Il tipo dei Nodi quindi dovrà essere uno dei *Node Type*.

Naturalmente più variabili dello stesso tipo sono presenti nello stesso progetto quibdi sarà sufficiente definirle una volta sola.

Solitamente i Nodi non sono molto complessi, così è possibile generarli automaticamente con il descrittore di dispositivi (*Devices Descriptors*) o dal compilatore/assembler.

L'interfaccia usata per monitorare il contenuto del MIB e permettere la sua descrizione è il MIBOCX. Questo è un OLE Control Extension (OCX) in accordo con Microsoft Windows. Il MIBOCX permette quindi di visualizzare la struttura come un albero, come ad esempio il File Manager di Windows. In questo caso però non ci sono driver, file o directory bensì Progetti, Nodi o Variabili.



Questo controllo OCX può essere chiamato da un linguaggio orientato agli oggetti come Visual Basic, C++ o Delphi.
L'OCX è usato all'interno di numerosi strumenti di VIGO incluso l'editor di MIB.

Seguendo lo standard Windows sarà possibile usare il tasto destro del Mouse per avere informazioni rilevanti riguardo il nodo, se posizionati su di un nodo, oppure riguardanti un progetto.

Quando VIGOSERV richiede informazioni al MIB usando un identificatore globale per un oggetto fisico, il MIB raccoglie tutte le informazioni necessarie riguardanti il particolare oggetto fisico di cui è stata fatta la riichiesta e poi li "restituisce " al VIGOSERV.

Quindi il MIB descrive come i dati sono strutturati, come differenti elementi sono in relazione fra loro, dove i dati sono i,mmagazzinati e chi può accedere ad essi. Esso perciò permette ad un impianto fisico di essere completamente descritto da un progetto (Project), in termini di dati, relazioni con strutture dati e locazioni dei dati stessi.

Una volta terminata la definizione dei dati, un sistema è in grado di acquisire automaticamente dati e distribuirli ai dispositivi di controllo come applicazioni Windows, Process computer, PC's, PLC, moduli I/O.

APPLICAZIONE

Il programma è una applicativo che deve essere usato da una ditta ad una fiera per dimostrare la possibilità di pilotare dei dispositivi fisici con un PC.

Data la natura del progetto quindi, requisiti e funzionalità non devono essere molto complesse ma è necessario solamente che l'applicativo renda l'idea delle potenzialità di questo sistema di controllo.

Quello che si dovrà consentire è la possibilità di selezione da parte dell'utente della modalità di comando dei dispositivi di I/O: nel nostro caso i Tank Sampler.

Le opzioni di scelta sono le seguenti:

1) regolazione della velocità dei tank dal selettore sul dispositivo: Al tank Sampler è collegato un selettore manuale a 12 posizioni che ne regola la frequenza di rotazione. Il selettore diventa attivo se un particolare flag della memoria viene impostato (sarà una variabile nella softwire list);

2) regolazione della velocità del tank dai cursori sul programma: Vengono abilitati dei cursori sul programma che regolano la velocità di rotazione dei motori;

3) Selezione della modalità programmata: Viene abilitato un form di input dove vengono inseriti i tempi di rotazione in senso regolare, riposo, rotazione in senso inverso.

Nel caso sia stata selezionata la modalità programmata, l'applicazione si dovrà occupare di gestire il funzionamento dei motori con i tempi impostati nella seguente modalità: rotazione del tank1 per il primo tempo impostato, pausa per il secondo, rotazione in senso invertito per il terzo, ancora pausa per il secondo tempo impostato dall'utente.

I due tank sampler funzionano su tempi indipendenti, quindi occorrerà impostarli entrambi.

È stata inclusa anche la possibilità di regolare la velocità dei motori sulla base di informazioni che provengono da dispositivi di input (nel nostro caso sensori di temperatura e di livello) connessi alla stessa rete.

Questo rende più chiaro come sia possibile automatizzare processi industriali grazie alla possibilità di agire anche sulla base di eventi che vengono registrati nell'ambiente in cui si opera.

Ecco che risulta possibile progettare dei sistemi di controllo in feedback. Si può ad esempio regolare il flusso di uscita da una cisterna sulla base del livello letto da un sensore, oppure attivare degli allarmi se la temperatura di un ambiente supera una certa soglia.

Nel programma infine viene inserita una maschera che visualizza in maniera chiara è semplice lo stato in cui si trova il sistema.

Le variabili che vengono monitorate sono:

-Velocità dei tank sampler

-Corrente assorbita dai tank

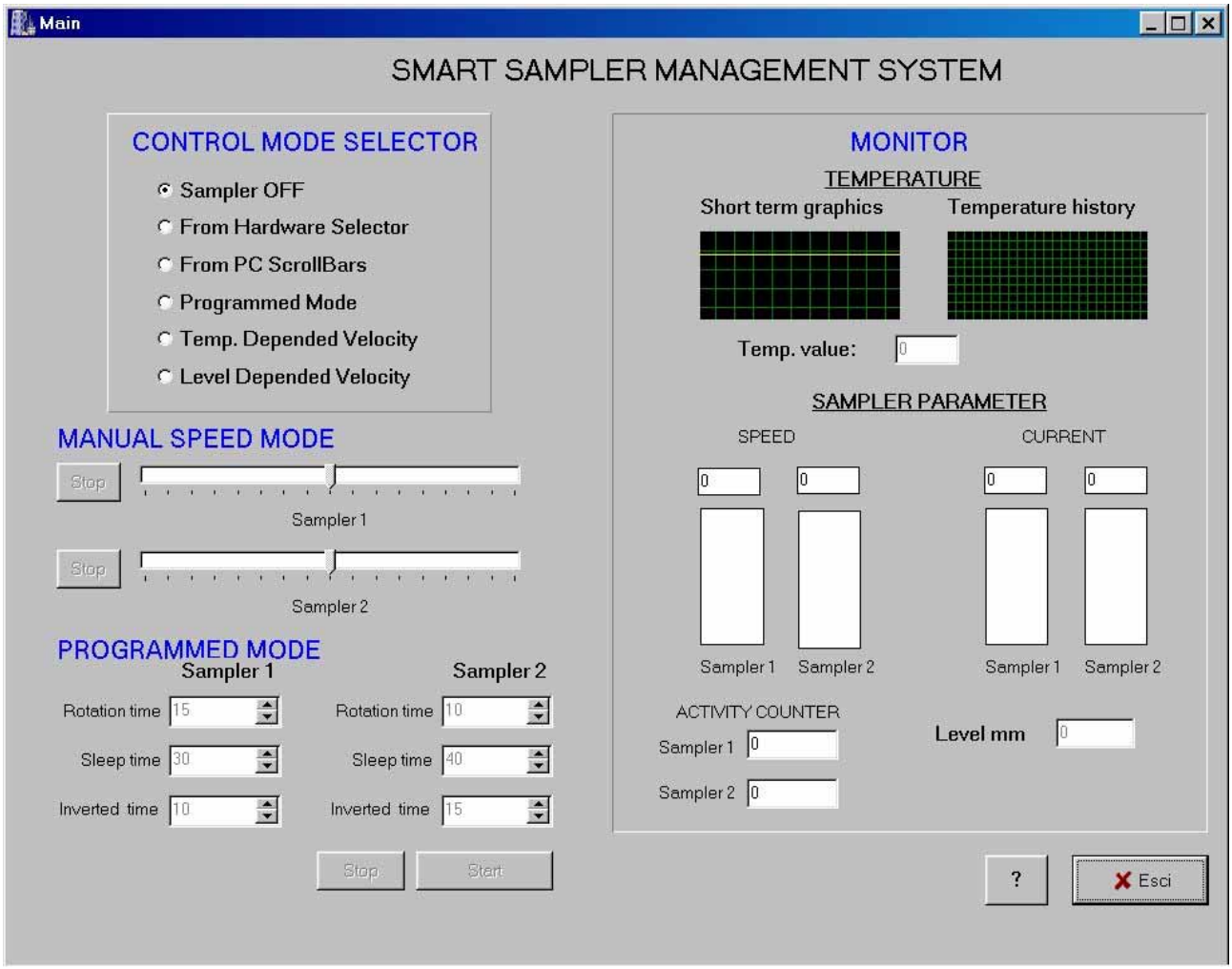
-Temperatura del sistema campionata dalla sonda con grafico a lungo e breve termine.

-Valori di livello misurati dalla sonda.

Questa maschera, che ha lo scopo di monitorare lo stato del sistema, resta attivo anche quando il controllo dei motori viene remotizzato verso macchine connesse al PC attraverso Internet. Questa parte verrà approfondita in seguito.

Vediamo ora qui di seguito alcune parti concettualmente interessanti del codice per il programma dimostrativo:

Foto della maschera iniziale



Routine principale (Main)

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <stdio.h>

#include "Main.h"
#include "VigoUtil.h"
#include "SamplerMng.h"


//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "CGAUGES"
#pragma link "PERFGRAP"
#pragma link "CSPIN"
#pragma resource "*.dfm"
TMainDlg *MainDlg;



//---------------------------------------------------------------------------
__fastcall TMainDlg::TMainDlg(TComponent* Owner)
    : TForm(Owner)
{
    //alla partenza del programma
    PrgAutoEnabled = false;

    RotTime[0] = RotTimeSmpl1;
    RotTime[1] = RotTimeSmpl2;
    SleepTime[0] = SleepTimeSmpl1;
    SleepTime[1] = SleepTimeSmpl2;
    InvertTime[0]= InvertTimeSmpl1;
    InvertTime[1]= InvertTimeSmpl2;

    WriteShort("Freq1", 0);
    WriteShort("Freq2", 0);
    WriteShort("Mode1", 1);
    WriteShort("Mode2", 1);

    ReadFloat("Temp", &BaseTemperature);
}

//---------------------------------------------------------------------------
void __fastcall TMainDlg::EsciClick(TObject *Sender)
{
    WriteShort("Freq1", 0);
    WriteShort("Freq2", 0);
    WriteShort("Mode1", 1);
    WriteShort("Mode2", 1);
    exit(0);
}

//---------------------------------------------------------------------------
void __fastcall TMainDlg::RBtnHWClick(TObject *Sender)
{
    if (RBtnSamplerOff->Checked == true) {
        //sampler Spenti
        BtnStopSmpl1->Enabled = false;
        Tank1Speed->Enabled   = false;
        BtnStopSmpl2->Enabled = false;
        Tank2Speed->Enabled   = false;
        BtnStopProgr->Enabled  = false;
        BtnStartProgr->Enabled  = false;

        RotTimeSmpl1->Enabled = false;
        RotTimeSmpl2->Enabled = false;
        SleepTimeSmpl1->Enabled = false;
        SleepTimeSmpl2->Enabled = false;
        InvertTimeSmpl1->Enabled = false;
        InvertTimeSmpl2->Enabled = false;

        // Chiamo le due funzione qui sotto
        BtnStopSmpl1Click(Sender);
        BtnStopSmpl2Click(Sender);
```

```
        // abilito mode  per tenerli fermi
        WriteByte("Mode1", 1);
        WriteByte("Mode2", 1);

    } else if (RBtnHW->Checked == true) {
        //sampler comandati da selettore
        // Blocca i motori e imposta su vigo lo switch per i selettori manuali
        // Disabilita il resto delle caselle di input
        BtnStopSmpl1->Enabled = false;
        Tank1Speed->Enabled   = false;
        BtnStopSmpl2->Enabled = false;
        Tank2Speed->Enabled   = false;
        BtnStopProgr->Enabled  = false;
        BtnStartProgr->Enabled  = false;

        RotTimeSmpl1->Enabled = false;
        RotTimeSmpl2->Enabled = false;
        SleepTimeSmpl1->Enabled = false;
        SleepTimeSmpl2->Enabled = false;
        InvertTimeSmpl1->Enabled = false;
        InvertTimeSmpl2->Enabled = false;

        // Chiamo le due funzione qui sotto
        BtnStopSmpl1Click(Sender);
        BtnStopSmpl2Click(Sender);

        // disabilito mode in modo da abilitare il selettore
        WriteByte("Mode1", 0);
        WriteByte("Mode2", 0);

    }else if (RBtnPC->Checked == true) {
        // Blocca i motori abilita le caselle di tank sampler speed e disabilita
        // quelle programmate
        BtnStopSmpl1->Enabled = true;
        Tank1Speed->Enabled   = true;
        BtnStopSmpl2->Enabled = true;
        Tank2Speed->Enabled   = true;
        BtnStopProgr->Enabled  = false;
        BtnStartProgr->Enabled  = false;

        RotTimeSmpl1->Enabled = false;
        RotTimeSmpl2->Enabled = false;
        SleepTimeSmpl1->Enabled = false;
        SleepTimeSmpl2->Enabled = false;
        InvertTimeSmpl1->Enabled = false;
        InvertTimeSmpl2->Enabled = false;

        // abilito mode
        WriteByte("Mode1", 1);
        WriteByte("Mode2", 1);

        // Chiamo le due funzione qui sotto
        BtnStopSmpl1Click(Sender);
        BtnStopSmpl2Click(Sender);

    }else if (RBtnProgram->Checked == true) {
        // Blocca i motori, disabilita i comandi diretti e aspetta il bottone send
        // (dopo il send controlla i campi
        // e se sono corretti inizia il cicli programmato)
        // funziona solo se si arriva da un radio bottom diverso (se si riseleziona
        // lo stesso non dovrebbe succedere niente
        BtnStopSmpl1->Enabled = false;
        Tank1Speed->Enabled   = false;
        BtnStopSmpl2->Enabled = false;
        Tank2Speed->Enabled   = false;

        RotTimeSmpl1->Enabled = true;
        RotTimeSmpl2->Enabled = true;
        SleepTimeSmpl1->Enabled = true;
        SleepTimeSmpl2->Enabled = true;
        InvertTimeSmpl1->Enabled = true;
        InvertTimeSmpl2->Enabled = true;

        BtnStartProgr->Enabled = true;
        BtnStopProgr->Enabled = false;
        PrgAutoEnabled = false;

        // abilito mode
```

```
        WriteByte("Mode1", 1);
        WriteByte("Mode2", 1);

        // Chiamo le due funzione qui sotto
        BtnStopSmpl1Click(Sender);
        BtnStopSmpl2Click(Sender);

        //abilito ciclo automatico
        //PrgAutoEnabled = true;
    }else if (RBtnLevelDepSpeed->Checked == true) {
        //velocità dipendente da sensore temperatura
        //salvo la temperatura di riferimento
        ReadFloat("Temp", &BaseTemperature);
    }else if (RBtnLevelDepSpeed->Checked == true) {
        //velocità dipendente da sensore livello
    }
}

//---------------------------------------------------------------------------
void __fastcall TMainDlg::CicloPnetTimer(TObject *Sender)
{
    //char code[100];
    long rc;
    float temp;
    static short OldSpeed1,OldSpeed2;
    short Spd1,Spd2,Curr1,Curr2;
    static int count;

    // ------------------------------------------
    // Parte relativa ai monitor
    // ------------------------------------------
    if (! count%2) {
        //eseguo solo ogni 2 passaggi
        short Level;
        ReadShort("Level", &Level);
        LevelValue->Text = Level;

        ReadShort("FieraParma:Tank1.MotorChnl.Motor_StS.Actul_Torque", &Curr1);
        ReadShort("FieraParma:Tank2.MotorChnl.Motor_StS.Actul_Torque", &Curr2);

        ReadShort("Freq1", &Spd1);
        ReadShort("Freq2", &Spd2);

        SpeedS1->Progress   = abs(Spd1);
        SpeedS2->Progress   = abs(Spd2);
        Speed1->Text        = Spd1;
        Speed2->Text        = Spd2;

        if (Spd1 != 0) {
            CurrentS1->Progress = Curr1;
            Current1->Text      = Curr1;
        } else {
            CurrentS1->Progress = 0;
            Current1->Text      = 0;
        }

        if (Spd2 != 0) {
            CurrentS2->Progress = Curr2;
            Current2->Text      = Curr2;
        } else {
            CurrentS2->Progress = 0;
            Current2->Text      = 0;
        }
    }

    //eseguo ad ogni passaggio
    ReadFloat("Temp", &temp);
    float t = (temp - 20.0f) * 15.0f;
    TempValue->Text = temp;
    ShortTermGraph->DataPoint(clYellow, abs((long)t)); //(color and plot value)
    ShortTermGraph->Update();
    // ------------------------------------------
    // Fine Monitor
    // ------------------------------------------

    if (RBtnHW->Checked == true) {
        SamplerCycleManager(0, 0);
        SamplerCycleManager(1, 0);
```

```
        return;
    } else if (RBtnPC->Checked == true) {
        // Pilotaggio secondo cursori
        short speed = Tank1Speed->Position; // * 100;
        //Tank1TextSpeed->Caption = Value;
        if (OldSpeed1  != speed) {
            OldSpeed1 = speed;
            if ((rc=WriteShort("Freq1", speed)) == 0 )
                ; //Tank1Err->Text="";
            else
                ; //Tank1Err->Text=GetLastPnetErrString(0);
        }
        speed = Tank2Speed->Position; // * 100;
        //Tank2TextSpeed->Caption = Value;
        if (OldSpeed2  != speed) {
            OldSpeed2 = speed;
            if ((rc=WriteShort("Freq2", speed)) == 0 )
                ; //Tank1Err->Text="";
            else
                ; //Tank1Err->Text=GetLastPnetErrString(0);
        }
        SamplerCycleManager(0, 0);
        SamplerCycleManager(1, 0);
    } else if (RBtnProgram->Checked == true) {
        //abilito ciclo automatico
        //PrgAutoEnabled = true;
        if (PrgAutoEnabled == true) {
            SamplerCycleManager(0, 1);
            SamplerCycleManager(1, 1);
        } else {
            WriteShort("Freq1", 0);
            WriteShort("Freq2", 0);
        }
    }else if (RBtnTempDepSpeed->Checked == true) {
        //velocità dipendente da sensore temperatura
        short speed = TemperatureSpeedCalc();
        //Tank1TextSpeed->Caption = Value;
        if (OldSpeed1  != speed) {
            OldSpeed1 = speed;
            WriteShort("Freq1", speed);
            WriteShort("Freq2", -speed);
        }
    }else if (RBtnLevelDepSpeed->Checked == true) {
        //velocità dipendente da sensore livello
        short speed = LevelSpeedCalc();
        if (OldSpeed1  != speed) {
            OldSpeed1 = speed;
            WriteShort("Freq1", speed);
            WriteShort("Freq2", -speed);
        }
    }
}

//---------------------------------------------------------------------------
void __fastcall TMainDlg::TempHistTimer(TObject *Sender)
{
    //ogni 1 secondo
    float temp;
    long cnt1,cnt2;
    ReadFloat("Temp", &temp);
    LongTermGraph->DataPoint(clBlue, temp*1.1); //(color and plot value)
    LongTermGraph->Update();

    ReadLong("Counter1",&cnt1);
    ReadLong("Counter2",&cnt2);
    Counter1->Text = cnt1;
    Counter2->Text = cnt2;
}

//---------------------------------------------------------------------------
void __fastcall TMainDlg::BtnStopSmpl1Click(TObject *Sender)
{
    Tank1Speed->Position = 0;
    WriteShort("Freq1", 0);
}

//---------------------------------------------------------------------------
void __fastcall TMainDlg::BtnStopSmpl2Click(TObject *Sender)
```

```
{
    Tank2Speed->Position = 0;
    WriteShort("Freq2", 0);
}

//---------------------------------------------------------------------------
short __fastcall TMainDlg::SpeedCalc(void)
{
    float temp;

    //if (SpeedCheckBox->Checked==true) {
        ReadFloat("Temp", &temp);
        return 1000 + (temp - BaseTemperature)*800;
    //}
    //return 500;
}

//---------------------------------------------------------------------------
short __fastcall TMainDlg::TemperatureSpeedCalc(void)
{
    float temp;
    ReadFloat("Temp", &temp);
    return 1000 + (temp - BaseTemperature)*800;
}

//---------------------------------------------------------------------------
short __fastcall TMainDlg::LevelSpeedCalc(void)
{
    short lev;
    ReadShort("Level", &lev);
    if (lev<0)
        lev=0;
    lev -=240;
    return lev * 30;
}

//---------------------------------------------------------------------------
void __fastcall TMainDlg::BtnStartProgrClick(TObject *Sender)
{
    PrgAutoEnabled = true;
    BtnStopProgr->Enabled=true;
    BtnStartProgr->Enabled=false;
}
//---------------------------------------------------------------------------

void __fastcall TMainDlg::BtnStopProgrClick(TObject *Sender)
{
    PrgAutoEnabled = false;
    BtnStopProgr->Enabled=false;
    BtnStartProgr->Enabled=true;
    WriteShort("Freq1", 0);
    WriteShort("Freq2", 0);
    SamplerCycleManager(0, 0);
    SamplerCycleManager(1, 0);
}
//---------------------------------------------------------------------------
```

Codice della Unit che contiene il sistema a stati finiti.

Per questa parte del codice si è scelto di implementare un automa a stati finiti in grado di funzionare in parallelo. Deve essere possibile infatti gestire indipendentemente il ciclo di funzionamento dei due tank sampler evitando per esempio una situazione nella quale il processo che attende lo scadere del timer di funzionamento di un tank, ignori il fatto che nel frattempo l'altro tank sampler doveva iniziare la rotazione. Un episodio di questo tipo infatti violerebbe le specifiche di progettazione.

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "SamplerMng.h"
#include "Main.h"
#include "VigoUtil.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)

#define TEMPO_DI_CICLO  200L

//---------------------------------------------------------------------------
void SamplerCycleManager(int Smplr, int Mode)
{
    static char *SmplrName[]={"Freq1","Freq2", 0};
    static short OldSpeed[2];
    static int Sts[2];
    static int Timer[2];

    short   NewSpeed;

    if ( (Smplr > 1) || (Smplr < 0) )
        return;

    if (Mode == 0) {
        //reset variabili
        Sts[Smplr] = 0;
        Timer[Smplr] = 0;
        OldSpeed[Smplr] = 0;
        return ;
    }

    Timer[Smplr] += TEMPO_DI_CICLO;

    switch (Sts[Smplr]) {
        case 0:
            NewSpeed = 2000;    //MainDlg->SpeedCalc();
            if (OldSpeed[Smplr] != NewSpeed) {
                WriteShort(SmplrName[Smplr], NewSpeed);
            }
            if (Timer[Smplr] >= MainDlg->RotTime[Smplr]->Value*1000) {
                WriteShort(SmplrName[Smplr], 0);
                Timer[Smplr] = 0;
                Sts[Smplr] = 1;
            }
            break;
        case 1:
            if (Timer[Smplr] >= MainDlg->SleepTime[Smplr]->Value*1000) {
                Timer[Smplr] = 0;
                Sts[Smplr] = 2;
            }
            break;
        case 2:
            NewSpeed = -2000;   //-MainDlg->SpeedCalc();
            if (OldSpeed[Smplr] != NewSpeed) {
                WriteShort(SmplrName[Smplr], NewSpeed);
            }
            if (Timer[Smplr] >= MainDlg->InvertTime[Smplr]->Value*1000) {
                WriteShort(SmplrName[Smplr], 0);
                Timer[Smplr] = 0;
                Sts[Smplr] = 3;
            }
            break;
```

```
        case 3:
            if (Timer[Smplr] >= MainDlg->SleepTime[Smplr]->Value*1000) {
                Timer[Smplr] = 0;
                Sts[Smplr] = 0;
            }
            break;
    }
}
```

CONTROLLO DEL SISTEMA DA MACCHINA REMOTA

Introduzione:

Oggi giorno, l'integrazione tra le varie applicazioni rappresenta una parte sempre piu' rilevante del lavoro svolto dai programmatori.

Lo stesso discorso puo' essere esteso tra hardware e software e il problema e' stato oggetto di studio anche nel caso dei sistemi basati su reti Pnet.

Uno dei metodi per l'integrazione di applicazioni diverse che e' diventato uno standard per le applicazioni che girano sotto Windows prede il nome di OLE (Object Linked Embedded) tecnology

L'utilizzo di Vigo con OLE consente di costruire applicazioni robuste per il controllo dei processi abbastanza facilmente. Ma l'integrazione di queste applicazioni con le tecnologie disponibili su Internet danno anche la possibilita' di realizzare funzioni di controllo di processo in modo remoto.

L'utilizzo di questa tecnologia diviene quindi fondamentale in quanto consente di ampliare notevolmente il le possibilita e il range di utilizzo dei sistemi P-Net based.

Analisi preliminare e studio di fattibilita':

I metodi che sono a disposizionedei programmatori per costruire applicazioni di controllo sono molti, ma si possono suddividere in due categorie:

- Applicazioni locali implementate in un ambiente di sviluppo come Delphi, C++ e VB. Le applicazioni basate su queste tecnologie pero' sono vincolate a girare su macchine il cui meccanismo di comunicazione e' quello definito dal programma stesso.

- WEB-SERVER: Programmi di comunicazione basati su HTTP le cui pagine contengono applet o script fatti in Java o VB. La gestione e il monitoraggio dei sistemi Pnet based puo' in questo caso essere effettuata da un qualsiasi PC connesso ad internet che sia dotato di browser.

Le applicazioni che si avvalgono di questa tecnologia presentano però degli svantaggi:

- tempi di comunicazione lunghi e variabili.

- necessita' di utilizzare dei sistemi per la protezione dei dati che attraversano la rete

Si potrebbe pensare che la disponibilita di sistemi di crittografia gia' fatti venga in aiuto al compito dei programmatori ma in realta' le cose non sono cosi semplici: i browsre Internet Explorer e Netscape sono divenuti di fatto uno standard. Sulla base delle legislazione Americana inoltre, la lunghezza massima delle chiavi di crittografia utilizzate per cifrare testo da esportare dall'america non puo' superare i 40bit : questo rende tutti i sistemi teoricamente vulnerabili. Alcuni browser contengono bug ed errori grazie ai quali diventa possibile intervenire nel sistema: questo rappresenta uno dei problemi principali.

Nel nostro caso pero' queste questioni non vengono considerate in maniera approfondita:Il nostro progetto ha solo lo scopo di dimostrare la fattibilita' del controllo remoto e i probliemi reltivi a sicurezze e a tempi di risposta, strettamente dipendenti dalle specifiche applicazioni, vengono lasciati ai casi reali.

DETERMINAZIONE DELL'AMBIENTE DI SVILUPPO E DEGLI STRUMENTI DA USARE

Essendo il progetto volto alla remotizzazione dei processi tramite Internet gli strumenti che useremo dovranno avere le seguenti caratteristiche:

1) essere in grado di gestire un Server Web.

2) supportare linguaggi in grado di manipolare oggetti ti tipo OLE.

Tra le soluzioni disponibili oggi giorno quelle che abbiamo valutato sono:

- Utilizzo di sistemi in grado di supportare WebServer con RMI Remote Method Invocation

- Utilizzo di Servlet (package Java utilizzabile per realizare applicazioni che girano su Web Server)

- Utilizzo di Server Web come (IIS e PWS) in grado di interpretare pagine che incudono comandi di scripting (VB o JavaScript) contenenti riferimenti ad oggetti di tipo OLE.

Tra queste abbiamo optato su Personal Web Server (pacchetto fornito insieme a WIN98) in quanto questo ambiente ci e' apparso il piu' indicato ai nostri scopo per ragioni di semplicita', disponobilita' e compatibilita' con i prodotti Microsoft.

CARATTERISTICHE DI PERSONAL WEB SERVER

Il Server Web PWS dovra' essere in grado di gestire pagine Web con le quali l'utente interagisce per modificare e monitorare il funzionampento del sistema PNet based.

La struttura dell'impianto sul quale operare corrispondera' quindi a quella definita nel MIB di VIGO. PWS rende inoltre possibile la convivenza sulla stessa macchina di Vigo, il programma di controllo locale e il sito Web.

Arrivati a questo punto abbiamo duvuto scegliere il modo in cui realizzarel'interazione tra pagine Web e oggetti di tipo OLE che permette di modificare i parametri del processo.

Le possibilita' che abbiamo valutato erano le seguenti:

- realizzare CGI

- realizzare Applet

- utilizzare la pagine ASP (supportate da PWS).

Tra queste abbiamo scelto le pagine ASP in quanto ci sono sembrate le piu' opportune ai nostri scopi.

I vantaggi principali di questa soluzione possono essere riassunti nel modo seguente:

- Le pagine asp sono completamente integrate con i file html.

- Sono facili da creare e non necessitano di compilazione.

- Sono orientate agli oggetti e usano componenti server ActiveX e all'interno di queste pagine possono essere inseriti sia i TAg Html che script (VB, Java, Delphi...) in grado di interagire con gli oggetti di Vigo.

PAGINE ASP

Una pagina asp e' un file di testo che contiene:

- Testo
- Tag HTML
- Comandi di Script che istruiscono il computer a fare qualcosa, come assegnare un valore ad una variabile.

Per creare i file ASP la procedura e' molto simile a quella che si utilizza per la creazione di pagine html. In questo caso pero' i file vengono salvati nel formato ASP. All'interno di questi file ci potranno essere dei comandi di Scripting che solo il personal web server sara' in grado di trasformare in html.

L'esecuzione di uno script invia una serie di comandi ad un motore di scripting (scripting engine), che lo interpreta. ASP contiene gli scripting engines per i linguaggi di scripting VBScript and JScript.

Abbiamo scelto Viasual Basic Script per la facilità e la somiglianza con il linguaggio Visual Basic. Il linguaggio di Scripting è necessario per crearer una classe in grado di interfacciare tutte le funzionalità di Vigo con le pagine ASP

VBScript comunica con applicazioni host tramite la tecnologia ActiveX(tm) Scripting, grazie alla quale nei browser e in altre applicazioni host non è necessario scrivere codice integrativo speciale per ciascun componente script.

Grazie alla tecnologia ActiveX Scripting i fornitori di linguaggi possono creare componenti run-time standard per le tecniche di script. Microsoft sta inoltre lavorando assieme a vari gruppi Internet per definire uno standard ActiveX Scripting che consenta di rendere intercambiabili i vari moduli di gestione per script. La tecnologia ActiveX Scripting viene utilizzata in Microsoft(r) Internet Explorer in Microsoft(r) Internet Information Server e in Persona Web Server.

CENNI SULLA SICUREZZA DEL SITO WEB

Per quanto riguarda la sicurezza, non sono state sviluppate grandi soluzioni in quanto non si trattava di comunicare dati relativi a carte di credito oppure messaggi diretti al ministero della difesa, bensì la modifica di dati di funzionamento di un impianto Demo.

In questo applicativo, la sicurezza del sito Web è affidata alla procedura di Login, all'utilizzo di una variabile di tipo session e dal fatto che tutti i comandi di scripting posti all'interno della pagina stessa non sono visibili da parte dell'utente che richiede la pagina. Infatti con opportuni TAG, è impossibile da parte dei client accedere alle intere pagine ASP, perché a loro verrà fornito solo codice html. Altro fattore a vantaggio della sicurezza consiste nel fatto che le operazioni che effettivamente andranno a modificare i valori dei vari dispositivi presenti nell'impianto sono state realizzate all'interno di un component server-side di Microsoft Visual Basic e quindi le chiamate vere e proprie non sono accessibili da Visitatori esterni.

L'utilizzo dei component server-side (da ora server component) sono utili per moltissime ragioni; la più importante nel nostro caso consiste nel fatto che protegge tutto il lavoro, dato che si tratta di una DLL (Dynamic Link Library) e che quindi non espone direttamente il codice sorgente.

Un server-components è una specie di controllo Active-X, ma invece di essere client side, come quelli sviluppati per Internet Explorer o lo stesso ambiente Visual Basic, sono usati in ambiente server. La differenza fondamentale tra client e server component è che l'ultimo ha l'assoluto divieto di esporre interfaccia utente, trattandosi di un componente che gira sulla macchina server, e che quindi non deve esporre finestre o pulsanti, in quanto queste verrebbero visualizzate sullo schermo della macchina server.

L'utilizzo di una variabile session, invece, permette di non abilitare nessun cliente a caricare direttamente la pagina più importante prima di aver fatto la procedura di verifica del sistema. Infine la procedura di Login e' abbastanza banale, infatti si andranno a verificare i campi dei form appena inseriti nella pagina da parte dell'utente, e si controlleranno con i valori predefiniti. Anche in questo caso la sicurezza sta nel fatto che tutti i comandi ASP non sono visibili dal Client.

Sicuramente in un applicazione reale sarebbe stato necessario realizzare un apparato di verifica degli accessi più sofisticato e robusto, realizzato con DataBase dove ad ogni persona sono associati i permessi di utilizzo dei vari controlli e dove le procedure di comunicazione di Login e Password fossero criptate.

IMPLEMENTAZIONE DEL SITO WEB

La realizzazione del sito Web consiste nel definire le varie pagine che saranno rese disponibili ai vari client. Le parti principali da realizzare consistono nella comunicazione con VIGO e nella parte relativa alla sicurezza dell'accesso.

La comunicazione consiste nel fare in modo che da pagine Web sia possibile modificare alcuni (non tutti) valori relativi a dispositivi presenti nell'impianto. Per fare questo si sono usati dei components server-side di Visual Basic. In questi server components sono state realizzate le procedure in grado di leggere e scrivere i dati all'interno dei vari dispositivi. Questo e' possibile perché VIGO, fa in modo che tutti gli impianti possano essere visti come un oggettto in cui le variabili rappresentano i vari dispositivi e sulle quali e' possibile effettuare delle operazioni di scrittura e lettura. All'interno del nostro server components, infatti, abbiamo importato un oggetto di tipo "Vigo.Std". Una volta importato l'oggetto, bastava richiamare le variabili associate ai vari dispositivi ed effettuare le operazioni necessarie (nel nostro caso operazioni di lettura e scrittura di dati).

In questo modo si e' creato l'oggetto InterfaceVigo.Dll. Con questo oggetto quindi si hanno a disposizione tutte le primitive in grado di agire sui vari dispositivi gestiti da VIGO, cosi' da poter far sembrare l'oggetto InterfaceVigo come una vera e propria interfaccia al programma stesso.

Una volta creato l'oggetto la pagina che effettivamente e' stata la piu' dispendiosa in termini di tempo per la sua realizzazione e' stata senza dubbio quella relativa al monitoraggio e modifica dei dati relativi dell'impianto. Il tutto e' stato realizzato unendo due pagine:

- La pagina di controllo, cioe' quella che fornisce solo i dati relativi all'impianto;

- La pagina di inserimento dei dati veri e propri.

Questa scelta e' stata dettata dal fatto che la pagina relativa alla sola visione dei dati poteva essere utilizzata anche da sola nel caso in cui si volesse accedere all'impianto solo da utente normale.

Questa pagina quindi fornirà una tabella dove vengono presentati tutti i dati relativi all'impianto il quale viene aggiornata ogni 5 sec. in modo da avere dei dati pressoché in tempo reale.

La pagina relativa all'inserimento dei dati da modificare invece si puo' accedere solo se in possessso della Logine e Password del SuperUser. In questo caso, e solo dopo aver inserito i valori per la verifica, si potra' accedere alla pagina in questione.

L'altra parte piu' dispendiosa dell'intero progetto, consisteva nella verifica dell'utente collegato. Per realizzare cio' ci siamo serviti di un Form di inserimento di dati e ad uno Script Visual Basic, in grado di verificare la reale corrispondenza tra i valori digitati dal client e quelli che realmente permettono di entrare come SuperUser. La verifica di questo, come appena spiegato, viene realizzata con uno script ASP. In questo modo il codice relativo alla verifica non viene visto dall'utente in quanto la pagina che viene spedita al Client e' html, che rappresenta la pagina ASP interpretata dal PWS.

L’utlima parte meno impegnativa ma altrettanto importante e’ stata quella di inserire all’interno delle varie pagine realizzate una variabile di tipo session. Questa variabile cosi’ definita e’ in grado di mantenere lo stesso valore per tutta la durata della sessione di collegamento, sia esso un valore impostato nella pagina iniziale che in quella corrente. In questo modo e’ stato possibile realizzare un ulteriore grado di sicurezza in quanto l’accesso alla pagina piu’ "delicata" e’ stato regolato verificando anche il valore di questa variabile. Infatti questa variabile conterra’ un valore di disabilitazione al caricamento della pagina interessata e questo valore verra’ modificato solo nel caso in cui vengano inseriti correttamente Login e Password.

Questo sistema sembra essere abbastanza efficente al fine di evitare accessi impropri degli utenti che tentano di evitare il riconoscimento tramite Login e Password e vogliono caricare direttamente la pagina in questione.

A questo punto è stato sufficiente richiamare l’oggetto creato in Visual Basic e utilizzare le funzionalità create ad Hoc per riuscire a modificare i dati veri e propri.

SULLA SICUREZZA

Arrivati a questo punto del lavoro, le potenzialita' del nostro applicativo vengono ampliate notevolmete: il controllo remoto di un processo da una qualsiasi macchina connessa ad Internet risulta essere utilissimo se non fondamentalein una vastissima gamma di situazioni.

Questi vantaggi pero' non sono privi di "side-effects" e occorre tener presente il rovescio della medaglia.

L'impiego di Internet per il controllo dei processi industriali comportail dover considerare tutta una serie di problemi relativi alla sicurezza.

Occorre infatti evitare nel modo piu' assoluto che eventuali malintenzionatiriescano a modificare o a leggere qualsiasi informazione inerente alfunzionamento del sistema e con Internet la cosa non e' cosi' semplice.

Diventa allora necessario il progetto di una politica di sicurezza adeguata.

Studio del problema

Analizziamo la situazione nella quale un'azienda interessata al controllo remoto potrebbe trovarsi:

- le persono si scambiano all'interno dell'azienda le chiavi di accesso alsistema di controllo. Il canale di comunicazioni delle chiavi quindi vieneassunto sicuro per ipotesi e i problemi relativi a possibili fughe di notizie oboicottaggi dentro l'azienda non rientrano nei nostri ambiti di studio.

- le persone che utilizzano il controllo devono poter accedere al sistema da una qualunque postazione connessa ad Internet. L'accesso deve essere consentito solo alle persono autorizzate dopo l'invio di una parola chiave

- occorre tenere traccia degli accessi al sistema di controllo memorizzando per ogni sessione di lavoro le seguenti informazioni:

  indentita' dell'utente,

  data e ora dell'utilizzo,

  operazioni effettuate e

  indirizzo IP della connessione.

Le problematiche relative alla sicurezza con le quali l'azienda potrebbe scontrarsi nella situazione descritta sopra possono allora essere riassunte nel modo seguente:

- necessita' di proteggere l'accesso alle pagine da persone non autorizzate: protezione delle sessione di lavoro
- necessita garantire l'integrita della chiave lungo il canale non sicuro
- fare in modo che chiunque entri in possesso del pacchetto contenente la
- chiave non riesca a farsi pasare per utente autorizzato e non tragga informazioni utili per scoprire la chiave
- fare in modo che chiunque entri in possesso dei pacchetti contenenti le informazioni relative al sistema (le pagine della sessione) non sia in grado di ricavarne da esse, informazioni utili

Il punto fondamentale allora consiste nel rendere sicura la connessione tra Client e Sever. Si potrebbe pensare di implementare un programma di comunicazione ad Hoc per le nostre esigenze, basato sul protocollo TCP/IP, ma in questo modo dovremmo forzare l'utente ad installare questa applicazione sulle macchine da usare. In questo modo pero' andremmo contro le specifiche di progettazione, che impongono l'utilizzo degli strumenti gia' disponibili nella maggior parte dei PC: Internet Explorer e Netscape.

La nostra soluzione allora deve essere cercata tra quelle gia' disponobili sul mercato e deve essere supportata dai browser piu' diffusi.

Il problema della sicurezza e' un argomento molto attuale. A tutt'oggi e' oggetto di molte ricerche, alimentate da prospettive di guadagno enormi e che vengono svolte dalle piu' grosse aziende informatiche.

In effetti basta solo pensare agli sviluppi dell commercio elettronico per avere un'idea della dimensione del business.

Questa situazione fa in modo che al giorno d'oggi sono disponibili molte soluzione alcune tra le quali sono divenute ormai uno standard riconosciuto di sicurezza.

Tra quest quella che sembra aver preso piede ultimamente e' il protocollo

SSL (Secure Socket Layer) il quale assicura uno standard di sicurezza adatto ai nostri scopi. Riportiamo il protocollo in appendice A

Nella nostra situazione questa soluzione garantisce un livello di sicurezza piu' che soddisfacente.

Nel nostro caso pero' pero' gli strumenti che abbiamo a disposizione non consentono l'utilizzo di questo protocollo.

Il programmi di comunicazione prodotti da Microsoft che supportano SSL sono Internet Information Server e Personal Web Server. IIS pero' gira solo su piattaforme Windows NT e non funziona su macchine Win9x, mentre la versione di PWS in grado di sostenere connessioni affidabili (SSL) e' quella per sistemi NT.

L'impiego di Win98 (sistema notoriamente non sicuro) riduce quindi notevolmente le possibilita' di utilizzo di programmi affidabili. A scopo puramente dimostrativo implementeremo nel nostro caso delle soluzioni che danno un minimo di sicurezza al nostro sistema. Tutto questo con la consapevolezza di indirizzare ogni azienda interessata al controllo remoto, verso soluzioni piu' efficenti.

Utilizzo di SCRIPT

Quello che cercheremo di fare consiste nel evitare il passaggio di alcune informazioni in chiaro nel pacchetto http. Precisamente le informazioni che vogliamo rendere incomprensibili al mondo esterno sono quelle relative alla chiave di accesso e ai dati del sistema da controllare.

Il linguaggio Html mette a disposizione dei programmatore i form: oggetti con i quali e' possibile scambiare informazioni tra client e server.

Occorera' allora catturare il contenuto dei form e criptarlo prima che venga inviato al server.

Questo e' possibile se al pulsante del form colleghiamo uno script. Vedremo in seguito la parte del codice relativa alla cattura del form:

Per garantire la privatezza delle informazione che dal client viaggiano verso il server utiizziamo un sistema crittografico a chiave pubblica. In questa maniera una volta generate le chiavi possiamo distribuire senza problemi l'algoritmo di codifica e la chiave pubblica ai vari client.

Questo viene fatto attraverso uno script che servira per rendere illeggibili le informazioni che vengono messe sulla rete. Per decifrare le informazioni che viaggiano sul server invece implementeremo l'algoritmo di codifica in visual-basic, la protezione e assicurata dal fatto che con PWS, il codice non e' visibile ai client, di conseguenza anche chiave privata e messaggio in chiaro.

La codifica che abbiamo usato e la RSA la cui descrizione viene riportata nell'appendice 2

Vediamo ora la parte relativa a form e a script di codifica e il codice per decifrare

FORM:

```
<FORM ACTION = "Decript.asp" METHOD = "post" NAME= FMOTORI style="text-align:
left">
<P>Motore A: <INPUT NAME = "MotA" SIZE = 5>
<P>Motore B: <INPUT NAME = "MotB" SIZE = 5>
<INPUT TYPE="hidden" NAME="Cript1">
<INPUT TYPE="hidden" NAME="Cript2">
<p><INPUT TYPE="button"   NAME="btnEnter1" Value="Invia" >
<INPUT TYPE=reset VALUE="Reset">
</FORM>
```

SCRIPT:

```
<SCRIPT LANGUAGE="VBScript">
<!--
Sub btnEnter1_OnClick
  Dim TheForm, St1,St2, i, x, y, d


  Set TheForm = Document.FMOTORI
  If IsNumeric(TheForm.MotA.Value) and IsNumeric(TheForm.MotB.Value) Then
     If TheForm.Mota.Value < 10000 and TheForm.MotB.Value < 10000  Then

        // implemento la codifica RSA sulla stringa dei codici ascii delle
        //  cifre relativa al numero inserito
        St1=Cstr(TheForm.MotA.Value)
        St2=Cstr(TheForm.MotB.Value)
```

```
        For i = 1 To Len(St1)
           x =  CInt(Mid(St1,i,1))
           cr = (x^3 mod 33)
           y = y + "-" + Cstr(cr)
        Next
        TheForm.Cript1.Value = y
        y=""
        For i = 1 To Len(St2)
           x =  CInt(Mid(St2,i,1))
           cr = (x^3 mod 33)
           y = y + "-" + Cstr(cr)
        Next
        TheForm.Cript2.Value = y
        TheForm.submit
      Else
        Msgbox "FMOTORI: Valori in ingresso troppo elevati"
      End If
  Else
        Msgbox "FMOTORI: Valori in ingresso non numerici."
  End if
End Sub
```

DECODIFICA FRASE CRIPTATA:

```
Dim M1,M2,C1,C2
M1 = Request.form("MotA")
M2 = Request.form("MotB")
C1 = Request.form("Cript1")
C2 = Request.form("Cript2")

Dim j,k, Cr1(5), Cr2(5), Chiaro1, Chiaro2
   Dim a, b
   j=0
   For i = 1 to Len(C1)
     If Mid(C1,i,1) = "-" then
        j=j+1
     Else
        Cr1(j)=Cr1(j)+Mid(C1,i,1)
     End if
   Next
   k=j
   j=0
   For i = 1 to Len(C2)
     If Mid(C2,i,1) = "-" then
        j=j+1
     Else
        Cr2(j)=Cr2(j)+Mid(C2,i,1)
     End if
   Next

   For i = 1 to k
     a = CLng(CInt(Cr1(i)))
     b= a^7
     b= b - (int(b / 33)*33)  'Calcolo il modulo per evitare overflow
     Chiaro1 = Chiaro1 + Cstr(b)
   Next
   For i = 1 to j
     a = CLng(CInt(Cr2(i)))
     b= a^7
     b= b - (int(b / 33)*33)  'Calcolo il modulo per evitare overflow
     Chiaro2 = Chiaro2 + Cstr(b)
   Next
Chiaro1 = CInt(Chiaro1)
Chiaro2 = CInt(Chiaro2)
```

APPENDICE A: PROTOCOLLO SSL VERSIONE 3.0

Indice:

3.0 Introduzione.

Questo documento illustra le specifiche della versione 3.0 del Secure Socket Layer protocol (SSL-v3.0) di Netscape Communication Corporation, (http://www.netscape.com) un protocollo che garantisce la privacy delle comunicazioni su Internet; esso permette infatti alle applicazioni client/server di comunicare in modo da prevenire le intrusioni, le manomissioni e le falsificazioni dei messaggi.

Il protocollo SSL provvede alla sicurezza del collegamento garantendo **tre funzionalità fondamentali:**

- **Privatezza del collegamento**. La crittografia è usata dopo un handshake iniziale per definire una chiave segreta. Per crittografare i dati è usata la crittografia simmetrica (e.g. DES, RC4, etc.).
- **Autenticazione.** L'identità nelle connessioni può essere autenticata usando la crittografia asimmetrica, o a chiave pubblica (per es. RSA, DSS, etc.). CosŒ i clients sono sicuri di comunicare con il corretto server, prevenendo ogni interposizione. E' prevista la certificazione sia del server che del client.
- **Affidabilità**. Il livello di trasporto include un check dell'integrità del messaggio basato su un apposito MAC (Message Authentication Code) che utilizza funzioni hash sicure (e.g. SHA, MD5, etc.). In tal modo si verifica che i dati spediti tra client e server non siano stati alterati durante la trasmissione.

Nota: Per una descrizione degli algoritmi di crittografia citati vedere l'**appendice.**

SSL è un protocollo aperto e non proprietario; è stato proposto da Netscape Communications al W3 Consortium come un possibile futuro approccio standard alla sicurezza per i browsers WWW e per i servers.

Lo scopo primario del SSL Protocol è fornire riserbo ed affidabilità alle comunicazioni. Il protocollo è composto da due strati (vedi figura 5): a livello più basso, interfacciato su di un protocollo di trasporto affidabile come il TCP, c'è il protocollo **SSL Record.** Questo è usato per l'incapsulamento dei vari protocolli di livello superiore. Sul protocollo SSL Record si interfaccia l'**SSL Handshake Protocol** che permette al server ed al client di autenticarsi a vicenda e di negoziare un algoritmo di crittografia e le relative chiavi prima che il livello applicazione trasmetta o riceva il suo primo byte. Un vantaggio del SSL è la sua **indipendenza dal protocollo di applicazione**: un protocollo di livello più alto può interfacciarsi sul protocollo SSL in modo trasparente.

3.1 Utilizzazione di SSL. INIZIO PAGINA

Per sfruttare la protezione offerta da SSL è **necessario** che un sito web disponga di un **server** in cui sia integrata **la tecnologia SSL.** Attualmente l'implementazione **SSLref v3.0b1** della Netscape Communications C., disponibile al sito http://home.netscape.com/newsref/std/sslref.html , è **mulipiattaforma** anche se sono differenziate le versioni per **Windows 95/NT** e per **Solaris**, in forma di pre-build file. Le versioni destinate alla vendita negli USA prevedono un più alto livello di sicurezza, mentre, a causa della legislazione USA in materia di export di algoritmi di crittografia, le versioni internazionali sono ristrette all'uso dell'algoritmo RC4 con chiavi di 40 bits (vedi appendice sugli algoritmi di crittografia), come implementato in Netscape Navigator. Da ricerche in Internet dell'ultimo minuto risulta che il server Netscape Commerce Server supporta SSLv2.0, mentre il nuovo **Netscape Enterprise** integra **SSLv3.0**; la politica di diffusione voluta dalla stessa Netscape Communication Inc. rende disponibili questi e tutti gli altri prodotti software a **studenti, universit€**

ed altre istituzioni non commerciali in modo **free al sito http://test-drive.netscape.com/edu_drive/index.html.** Anche il **client** deve supportare SSL per poter stabilire una connessione sicura con un server SSL: Netscape Navigator lo supporta dalla versione 0.93.

Per Unix e Windows 3.1/95/NT esiste la versione **SSLLeay 0.6.0** che implementa **SSLv2.0**, free sia per uso privato che commerciale, disponibile in forma di pre-built DLL e di codice sorgente VisualC++ : include una implementazione di DES tra le più veloci, oltre agli algoritmi di crittografia più comuni, IDEA, RC4, RSA, e MD5 (vedi appendice).

Inoltre è disponibile **Secure HTTP e SSL Toolkit** commercializzato da Terisa: "http://www.terisa.com/prod/prod.html" .

Web server sicuri di IBM sono distribuiti per le maggiori piattaforme al sito: http://www.ics.raleigh.ibm.com/.

Con questi mezzi è possibile usare, per esempio, **https,** cioè http con SSL, e scambiare informazioni con un client per mezzo di https. Poichè **http+SSL** e **http** sono differenti protocolli ed usano porte diverse, lo stesso sistema server può far girare contemporaneamente sia il server **http+SSL** che quello **http.** Ciò significa che un server può offrire alcune informazioni a tutti gli utenti senza sicurezza, ed altre solo in modo sicuro: ad esempio un catalogo può essere "non sicuro" mentre gli ordini ed i pagamenti devono essere protetti.

Riferendoci sempre a Netscape Navigator, illustriamo le modalità con cui è possibile usare SSL, riportando come esempio l'esperienza del collegamento alla pagina http://www.cyberpi.com/ dove esiste un link ad una pagina di ordinazione dei servers prodotti da Cyber Presence, protetta da SSL. Ribadiamo a tal fine che il browser puø essere qualunque, purch‰ supporti il protocollo SSL e, quindi, il nuovo metodo di accesso URL **https** per connessioni con un server che usa SSL. **Https** è il protocollo che si ottiene interfacciando http su SSL: si dovrà usare `"https://"` per gli URL http con SSL, mentre si continuerà ad usare `"http://"` per gli URL senza SSL. La porta di default per `"https"` è la numero 443, come stabilito dalla Internet Assigned Numbers Authority.
Per default una "security colorbar" appare in alto a destra in ogni finestra di Netscape Navigator: se la barra è grigia allora il documento che stiamo ricevendo non è "sicuro", mentre se la barra è blu, il documento corrente è sicuro, cioè trasferito in modo protetto dalla crittografia. Una icona all'angolo in basso a sinistra mostra la stessa situazione: se è visibile una chiave rotta su sfondo grigio allora la connessione non è sicura, mentre se la chiave è intera su sfondo blu la connessione è sicura. Inoltre la voce "Document Information" nel menù "File" mostra una dialog box che contiene informazioni sullo stato della connessione: il livello di crittografia usato, che per adesso, in accordo alle leggi sull' export degli USA, è ristretto all'uso di chiavi di soli 40 bit con algoritmo RC4; l'identità del server, ricavata dal suo certificato.

3.2 Obiettivi. INIZIO_PAGINA

Gli scopi del Protocollo SSL v3.0, in ordine di priorità, sono:

- **Sicurezza della crittografia.** SSL stabilisce un collegamento sicuro tra due sistemi.
- **Interoperabilità.** Programmatori di diverse organizzazioni dovrebbero essere in grado di sviluppare applicazioni utilizzando SSL 3.0, scambiandosi parametri della crittografia senza necessità di conoscere il codice l'uno dell'altro.
- **Ampliamento.** SSL cerca di fornire una struttura dentro la quale i nuovi metodi di crittografia a chiave pubblica e non, possano essere incorporati se necessario. Questo fa sì che anche altri due aspetti importanti vengano soddisfatti: prevenire il bisogno di creare un nuovo protocollo ed evitare la necessità di implementare una nuova security library.

- **Efficienza.** Le operazioni di crittografia tendono a essere molto laboriose per la CPU, particolarmente le operazioni con le chiavi pubbliche. Per questa ragione l' SSL ha incorporato uno schema di session caching opzionale per ridurre il numero di collegamenti che hanno bisogno di essere stabiliti ex-novo. Particolare attenzione è stata posta nel ridurre l'attività sulla rete.

3.3 Il protocollo SSL: soluzioni tecniche. INIZIO_PAGINA

Il protocollo SSL è un protocollo a livelli; ad ogni livello i messaggi possono includere campi per la lunghezza, la descrizione ed il contenuto. SSL procede sui messaggi da trasmettere nel modo seguente: **li frammenta in blocchi adeguati, eventualmente comprime i dati, applica un MAC, crittografa e infine trasmette il risultato. I dati ricevuti, viceversa, sono messi in chiaro, verificati, scompattati e riassemblati e, quindi, consegnati ai client di livello superiore.**

In particolare SSL prevede un **handshake di "sicurezza"** usato per iniziare la connessione TCP/IP: il risultato di tale handshake è la contrattazione da parte del client e del server del livello di sicurezza da usare ed il completamento delle autenticazioni necessarie alla connessione. Quindi SSL procede alla **cifratura** (e/o alla messa in chiaro) della sequenza di bytes del protocollo applicazione usato, ad esempio HTTP, SHTTP, NNTP o Telnet. Ciò significa che tutte le informazioni sia nell'HTTP request che nell'HTTP response sono completamente crittografate, incluso l'URL richiesto dal client, qualsiasi contenuto di forms compilati (**quindi anche eventuali numeri di carte di credito...**), ogni informazione sulle autorizzazioni all'accesso come usernames e passwords, e tutti i dati risposti dal server al client.

3.3.1 Stato della sessione e della connessione. INIZIO_PAGINA

E' responsabilità dell'SSL Handshake protocol coordinare gli stati del client e del server, permettendo così ai sistemi operativi di ognuno di operare in modo consistente, nonostante lo stato non sia esattamente parallelo. Logicamente lo stato è rappresentato in modo doppio: una volta come lo stato *operativo* corrente, e, durante la fase di handshake, come lo stato in *attesa;* inoltre sono mantenuti separati gli stati di lettura da quelli di scrittura. Quando il client o il server riceve un messaggio di **change cipher spec**, (vedi paragrafo 3.3.3) esso copia lo stato di lettura in attesa nello stato di lettura corrente. Quando il client o il server spedisce un messaggio di **change cipher spec,** copia lo stato in attesa di scrittura nello stato corrente di scrittura. Quando la negoziazione iniziale (handshake) è completa il client ed il server si scambiano il messaggio di **change cipher spec** ed iniziano a comunicare usando la nuova cipher spec stabilita nel modo illustrato sopra. Una sessione SSL può includere più di una connessione sicura; inoltre le applicazioni possono avere più sessioni simultanee.

Lo **stato della sessione** include i seguenti elementi:

- **session identifier:** è una sequenza arbitraria di bytes scelta dal server per identificare un stato attivo della sessione, o comunque riesumabile.
- **peer certificate:** certificato X.509 del peer. Può essere nullo.
- **compression method:** l'algoritmo usato per comprimere i dati prima della crittografia.
- **cipher spec:** specifica l'algoritmo di crittografia bulk (ad es. null, DES, etc.) e l'algoritmo MAC (per es. MD5 o SHA). Stabilisce inoltre dei parametri della crittografia come l' *hash_size* (vedi paragrafo 3.3.2.3.1 per una definizione formale).
- **master secret:** 48 bytes segreti, condivisi dal client e dal server.
- **is resumable:** è un flag che indica se la sessione può essere usata per iniziare nuove connessioni.

Lo **stato della connessione** include i seguenti elementi:

- **server and client random:** sequenza di bytes scelti dal server e dal client per ogni connessione.
- **server write MAC secret:** la sequenza segreta usata nelle operazioni MAC sui dati scritti dal server.
- **client write MAC secret:** la sequenza segreta usata nelle operazioni MAC sui dati scritti dal client.
- **server write key:** la chiave per i dati cifrati dal client e messi in chiaro dal server.
- **initialization vectors:** se è usato un blocco cifrato in modo CBC (vedi appendice), allora per ogni chiave è mantenuto un vettore d'inizializzazione (IV). Questo campo è inizializzato dal SSL handshake protocol. Poi il codice cifrato finale di ogni record è salvato per essere usato con i record seguenti.
- **sequence numbers:** ognuna delle due parti comunicanti mantiene separati numeri di sequenza per i messaggi spediti e ricevuti per ogni connessione. Quando uno dei due manda o riceve un messaggio di **change cipher spec,** il numero appropriato della sequenza è posto a zero. I numeri di sequenza sono espressi da 64 bits e non possono quindi eccedere 2^{64}-1.

3.3.2 Record layer. INIZIO PAGINA

Il Record layer del protocollo SSL riceve i dati dai livelli superiori senza interpretarli, in forma di blocchi non vuoti di dimensione arbitratria.

3.3.2.1 Frammentazione.

Il Record layer frammenta i blocchi di informazioni in SSLPlaintext records, di dimensione massima di 2^{14} bytes, la cui struttura è illustrata di seguito. Le divisioni del messaggio del client possono non essere rispettate nel record layer: per esempio messaggi multipli del client dello stesso ContentType possono essere fusi in un unico SSLPlaintext record.

Per descrivere le strutture dati useremo una pseudo-codifica molto simile al linguaggio 'C' con l'aggiunta di note esplicative laddove ci allontaneremo dallo standard del 'C'.

Il SSLPlaintext record ha la struttura seguente:

```
struct {
uint8 major, minor; /* uint8 indica un unsigned int di 8 bit
} ProtocolVersion;

enum {
change_cipher_spec(20), alert(21), handshake(22), application_data(23), (255)
} ContentType;

/* N.B. Il tipo enum specifica una lista di costanti seguite da un numero tra
parentesi che indica la numerazione assegnata alle costanti. Si assume che la
dimensione del tipo enum sia quella sufficiente ad esprimere il numero d'ordine
più alto assegnato ad una costante. Un numero tra parentesi non preceduto da una
costante (255 per esempio) è usato per forzare la dimensione del tipo enumerato
ad essere tale da poter esprimere almeno tale numero. Nel caso in esame la
presenza di (255) comporta che il tipo ContentType sarà rappresentato con 1
byte, necessario a rappresentare il numero 255, anche se in realtà un dato di
tipo ContentType può assumere solo 4 valori: 20, 21, 22 e 23 */

struct {
ContentType type;
ProtocolVersion version;
uint16 length;
```

```
opaque fragment[SSLPlaintext.length];
} SSLPlaintext;

/* N.B. Con il tipo opaque indicheremo un byte che SSL non può risolvere
ulteriormente, né come numero, né come carattere, in quanto tale byte fa parte
del flusso di dati di livello applicazione trattato in modo trasparente dal SSL.
*/
```

I campi della struttura SSLPlaintext hanno il seguente significato:

- **type:** il protocollo di livello più alto usato per processare il frammento incluso.
- **version:** la versione del protocollo impiegato, nel nostro caso SSL v3.0.
- **lenght:** la lunghezza in bytes del frammento SSLPlaintext che segue. Non può eccedere 2^{14}.
- **fragment:** i dati provenienti dal livello applicazione. Questi dati sono trasparenti e considerati come un blocco indipendente per essere poi trattati dal protocollo di livello più alto specificato nel campo type.

Da notare che dati di differenti ContentTypes possono essere alternati. I dati dell'applicazione hanno in genere la più bassa priorità di trasmissione rispetto agli altri ContentTypes.

3.3.2.2 Compressione e Decompressione del Record. INIZIO_PAGINA

Tutti i records sono compressi tramite un algoritmo definito nello stato corrente della sessione. C'è sempre un algoritmo di compressione attivo anche se all'inizio è definito come `CompressionMethod.null`. L'algoritmo di compressione trasforma una struttura SSLPlaintext in una SSLCompressed structure. Le funzioni di compressione cancellano le loro informazioni di stato quando la CipherSpec è rimpiazzata.

Nota: la CipherSpec è parte dello stato della sessione descritto nel paragrafo 3.3.1.

La compressione deve essere senza perdita e non può aumentare la lunghezza del contenuto più di 1024 bytes. Se la decompressione incontra un `SSLCompressed.fragment` che scompattato fosse lungo più di 2^{14} bytes, deve segnalare un fatale `decompression_failure alert` (paragrafo 3.3.4.2).

La struttura corrispondente è la seguente:

```
struct {
ContentType type; /* come SSLPlaintext.type */
ProtocolVersion version; /* come SSLPlaintext.version*/
uint16 length;
opaque fragment[SSLCompressed.length];
} SSLCompressed;
```

Significato dei campi:

- **length:** la lunghezza in bytes del seguente frammento SSLCompressed. La lunghezza non deve eccedere 2^{14} + 1024.
- **fragment:** la forma compressa del SSLPlaintext.fragment.

Note: l'operazione `CompressionMethod.null` è un'operazione d'identità che non altera alcun campo. La funzione di decompressione è responsabile del controllo di eventuali buffer overflows interni.

3.3.2.3 Protezione utile del Record e CipherSpec. INIZIO_PAGINA

Tutti i records sono protetti usando gli algoritmi di crittografia e MAC definiti nel corrente CipherSpec. C'è sempre un CipherSpec attivo, anche se inizialmente è posto uguale a `SSL_NULL_WITH_NULL_NULL` che non fornirebbe alcuna sicurezza.

Quando l'handshake è completato, le due parti hanno condiviso le stringhe segrete che sono usate per cifrare i records e computare i codici di autenticazione con chiave dei messaggi , (MACs) sulla base dei loro contenuti. Le tecniche usate per le operazioni di crittografia e di MAC sono stabilite nel `CipherSpec.cipher_type;` tali tecniche trasformano una struttura SSLCompressed in una SSLCiphertext. La decifratura inverte tale trasformazione. Le trasmissioni includono anche un **numero di sequenza**, in modo che i messaggi persi, alterati o intrusi siano rilevabili.

La struttura SSLCiphertext è la seguente:

```
struct {
ContentType type;
ProtocolVersion version;
uint16 length;
select (CipherSpec.cipher_type) {
case stream: GenericStreamCipher;
case block: GenericBlockCipher;
} fragment;
} SSLCiphertext;
```

Il significato dei campi è il seguente:

- **type:** come il `SSLCompressed.type`.
- **version:** come il `SSLCompressed.version`.
- **lenght:** la lunghezza in bytes del seguente `SSLCiphertext.fragment`. Non può eccedere 2^{14} + 2048.
- **fragment:** la forma cifrata del `SSLCompressed.fragment,` incluso il MAC; sono distinte le due strutture diverse a seconda che sia stato scelto l'uso di un algoritmo di crittografia che lavora su tutto lo stream di bit dei dati, `GenericStreamCipher,` o a blocchi, `GenericBlockCipher`, che defineremo nei prossimi due paragrafi.

3.3.2.3.1 Cifratura standard con algoritmi che lavorano sull'intero stream. INIZIO_PAGINA

Gli algoritmi stream ciphers, incluso il `BulkCipherAlgorithm.null`, convertono le strutture `SSLCompressed.fragment` in stream `SSLCiphertext.fragment` e viceversa, usando la struttura:

```
stream-ciphered struct {
opaque content[SSLCompressed.length];
opaque MAC[CipherSpec.hash_size];
} GenericStreamCipher;
```

`N.B. Le operazioni di crittografia da compiere sulle strutture sono designate tramite uno dei seguenti prefissi: digitally-signed, stream-ciphered, block-ciphered, e public-key-encrypted. Tali prefissi sono posti prima della descrizione del tipo della struttura; nel caso della struttura GenericStreamCipher sopra definita, abbiamo preposto "stream-ciphered", indicando cosŒ che essa verr€ cifrata con un algoritmo che processa l'intero flusso dei dati. Analogamente per la struttura GenericBlockCipher useremo il prefisso "block-ciphered".`

Il **MAC** è così generato:

hash (MAC_write_secret + pad_2 + hash (MAC_write_secret + pad_1 + seq_num +length + content));

ove:

- " + " denota la concatenazione.
- pad_1: è il carattere 0x36 ripetuto 48 volte per MD5 o 40 volte per il SHA.
- pad_2: il carattere 0x5c ripetuto lo stesso numero di volte del precedente.
- seq_num: il numero di sequenza del messaggio.
- hash: l'algoritmo hash definito nella CipherSpec.

Da notare che il MAC è calcolato prima della cifratura: la stream cipher crittografa l'intero blocco incluso il MAC. Se il campo `CipherSuite, che definiremo più avanti è SSL_NULL_WITH_NULL_NULL`, la cifratura lascia inalterata la struttura: i dati non sono cifrati ed il MAC vale zero, ovvero non è usato MAC. Infine la `SSLCiphertext.lenght = SSLCompressed.lenght + CipherSpec.hash_size.`

3.3.2.3.2 CBC block cipher INIZIO_PAGINA

Gli algoritmi block ciphers, come RC2 o DES, trasformano la struttura `SSLCompressed.fragment` in un blocco `SSLCiphertext.fragment tramite la seguente struttura:`

```
block-ciphered struct {
opaque content[SSLCompressed.length];
opaque MAC[CipherSpec.hash_size];
uint8 padding[GenericBlockCipher.padding_length];
uint8 padding_lenght;
} GenericBlockCipher;
```

dove:

- il MAC è generato come descritto nella sezione 3.3.2.3.1 .
- `padding:` riempitivo aggiunto per forzare la lunghezza del plaintext ad essere un multiplo della lunghezza dei blocchi cifrati dall'algoritmo di crittografia.
- `padding_lenght:` la lunghezza del padding deve essere minore di quella del blocco cifrato e può essere zero; questo parametro deve essere tale che la dimensione totale della struttura GenericBlockCipher sia un multiplo della lunghezza dei blocchi cifrati dall'algoritmo usato.

Nota: la lunghezza dei dati cifrati è complessivamente la somma della `SSLCompressed.lenght, CipherSpec.hash_size, padding_lenght` più uno.

3.3.3 Il protocollo Change cipher spec. INIZIO_PAGINA

Questo protocollo è finalizzato a segnalare le transizioni nelle strategie di crittografia. Il protocollo consiste di un singolo messaggio, che è cifrato e compresso secondo il `Current` (e non il `pending`) `CipherSpec`. Il messaggio consiste di un singolo byte di valore uno (ciò spiega la lunghezza dei dati cifrati!).

```
struct {
enum { change_cipher_spec(1), (255) } type;
} ChangeCipherSpec;
```

Il messaggio di **change cipher spec** è mandato sia dal client che dal server per notificare all'altro che da quel momento in poi i records saranno protetti usando le nuove, appena negoziate

CipherSpec e chiavi. La ricezione di questi messaggi causa al ricevente la copia del *pending read state* nel *current read state*: infatti sia il server che il client mantengono separato questo doppio stato. Quando il client o il server manda un messaggio di **change cipher spec,** esso copia il *pending write state* nel *current write state*.
Il client manda allora un messaggio di **change cipher spec** seguente i messaggi di handshake **key exchange** and **certificate verify,** ed il server ne manda uno dopo aver processato con successo il messaggio di **key exchange** ricevuto dal client. Quando l'handshake è completato il client ed il server si scambiano i messaggi di **change cipher spec** visti sopra ed iniziano a comunicare usando le nuove proprietà appena concordate. Un messaggio di **change cipher spec** inaspettato provoca un `unexpected_message` **alert** (vedi sezione 3.3.4.2). Quando si riesuma una sessione, il messaggio di **change cipher spec** è spedito subito dopo il messaggio di `hello`.

3.3.4 Alert protocol. INIZIO_PAGINA

Uno dei tipi di contenuto supportato dal livello SSL Record è l' ***alert***. I messaggi di ***alert*** comunicano la severità del messaggio e una descrizione dell'allerta: un messaggio con livello *fatal* si risolve con la immediata terminazione della connessione. In questo caso le altre connessioni nella stessa sessione possono essere continuate, ma l'identificativo della sessione deve essere invalidato in modo che non siano stabilite nuove connessioni. Come gli altri messaggi anche quelli di ***alert*** sono cifrati e compressi come specificato nello stato corrente della connessione.

Le strutture usate sono le seguenti:

```
enum { warning(1), fatal(2), (255) } AlertLevel;

enum {
close_notify(0), unexpected_message(10), bad_record_mac(20),
decompression_failure(30), handshake_failure(40), no_certificate(41),
bad_certificate(42), unsupported_certificate(43), certificate_revoked(44),
certificate_expired(45), certificate_unknown(46), illegal_parameter(47), (255)
} AlertDescription;


struct {
AlertLevel level;
AlertDescription description;
} Alert;
```

3.3.4.1 Allerta di chiusura. INIZIO_PAGINA

Il client e il server devono comunicarsi che la connessione sta terminando per evitare un attacco. Entrambi possono iniziare lo scambio dei messaggi di chiusura. In particolare il messaggio `close_notify` che notifica al ricevente che il mittente non trasmetterà più su quella connessione. La sessione diviene non riesumabile se qualche connessione è terminata senza il dovuto messaggio `close_notify` con livello `warning`.

3.3.4.2 Allerta per errori. INIZIO_PAGINA

Il trattamento degli errori nell'SSL Handshake protocol è molto semplice. Quando un errore è rivelato, la parte che lo ha rivelato manda un messaggio all'altra; dopo la trasmissione/ricezione di un messaggio di **fatal alert,** entrambe le parti immediatamente chiudono la connessione. Sia il server che il client sono obbligati a cancellare ogni identificatore di sessione, chiave, e altri segreti associati con la connessione fallita.

Sono definiti i seguenti messaggi di allerta per errore:

- `unexpected_message:` un messaggio inappropriato è stato ricevuto. Questo tipo di allerta è sempre fatale.
- `bad_record_mac`: questa allerta è spedita quando un record è ricevuto con un errato MAC; questo messaggio è sempre fatale.
- `decompression_failure`: la funzione di decompressione ha ricevuto un errato input, ad esempio dati che si espandono oltre la lunghezza loro assegnata; questo messaggio è sempre fatale.
- `handshake_failure`: indica che il mittente è incapace di negoziare un set accettabile di parametri di sicurezza tra quelli possibili; questo messaggio è sempre fatale.
- `no_certificate`: può essere mandato in risposta ad un **certificate request** se non è disponibile una certificazione appropriata.
- `bad_certificate`: in caso di certificazione errata, contenente una firma che non è verificata correttamente, etc.
- `unsupported_certicate`: se una certificazione è di un tipo non supportato.
- `certificate_revoked`: una certificazione è stata revocata dal suo firmatario.
- `certificate_expired`: una certificazione è scaduta o attualmente non valida.
- `certificate_unknown`: qualche altro non specificato problema è sorto nel processare la certificazione.
- `illegal_parameter`: un campo del handshake è di dimensione errata o inconsistente con altri campi; questo messaggio è sempre fatale.

3.3.5 Handshake protocol overview. INIZIO PAGINA

I parametri per la crittografia che compongono lo stato della sessione sono prodotti dal **SSL Handshake Protocol**, che opera interfacciandosi in basso con il **SSL Record Layer**. Quando un client ed un server SSL iniziano a comunicare, **concordano sulla versione del protocollo, scelgono gli algoritmi di crittografia, opzionalmente si autenticano l'un l'altro, e usano la crittografia a chiave pubblica per generare dati segreti condivisi.** Questi processi sono eseguiti nel handshake protocol, che può essere descritto cosi:

il **client** spedisce un messaggio di **client hello** al quale il **server** deve rispondere con un messaggio di **server hello,** altrimenti un errore fatale si verifica e la connessione fallisce. I messaggi **client hello** e **server hello** sono usati per stabilire le prestazioni di sicurezza ottenibili fra client e server; questi messaggi stabiliscono i seguenti attributi: `protocol version, session ID, cipher suite e compression method`. Inoltre, due valori casuali sono generati e scambiati: `ClientHello.random e ServerHello.random`.

Di seguito al messaggio di hello il server spedisce il suo certificato se questo deve essere autenticato; inoltre un messaggio di **server key exchange** può essere spedito, se è richiesto, ad esempio se il server non ha certificato, o se il suo certificato è solo per la firma. Se il server è autenticato, può richiedere un certificato dal client se ciò è in accordo con la cipher suite scelta.

Adesso il server manderà il messaggio di **server hello done**, indicando che la fase hello-message dell' handshake è completa, ed attenderà una risposta dal client.

Se il server ha mandato un messaggio di **certificate request**, il client deve spedire o un **certificate message** o una **no certificate alert**. Il messaggio di **client key exchange** viene mandato adesso, il cui contenuto dipende dall'algoritmo a chiave pubblica selezionato con il client hello ed il server

hello. Se il client ha spedito un certificato con abilitazione alla firma, allora un messaggio di **certificate verify è** spedito per verificare esplicitamente il certificato.

A questo punto un messaggio di **change cipher spec** è mandato dal client, che copia il *pending* `Cipher Spec` nel *current* `Cipher Spec`. Il client manda immediatamente il messaggio **finished** usando i nuovi algoritmi, chiavi e stringhe segrete concordate . Il server risponde con un messaggio di **change cipher spec**, copia il *pending Cipher Spec* nel *current Cipher Spec*, e spedisce il suo messaggio **finished** in accordo con la nuova `Cipher Spec`. Adesso la fase di handshake è completata ed il client ed il server possono cominciare a scambiare dati del livello applicazione.

Se il client ed il server decidono di riabilitare una precedente sessione o di duplicarne una esistente invece di negoziare di nuovo i parametri di sicurezza, il flusso dei messaggi è il seguente:

Il client manda un **client hello** usando il `Session ID` della sessione da riesumare; il server allora controlla la sua session cache per trovare una corrispondenza: se questa è trovata il server manda un **server hello** con lo stesso `Session ID.` Adesso sia il client che il server devono mandare un messaggio di **change cipher spec** e procedere direttamente fino al messaggio di **finished.** Una volta che il ripristino è completo, il client ed il server possono iniziare a scambiare dati di livello applicazione. Se il server non trova una corrispondenza di `Session ID` nella propria session cache, allora il server genera un nuovo `Session ID,` e verrà eseguito un handshake completo.

Il significato dei messaggi sopra citati è brevemente esposto nel paragrafo successivo.

3.3.6 Messaggi di Handshake. INIZIO_PAGINA

Il protocollo SSL Handshake è usato per negoziare gli attributi per la sicurezza di una sessione; i messaggi di handshake sono forniti al SSL Record Layer, in cui sono incapsulati dentro una o più strutture SSLPlaintext, a loro volta processate e trasmesse secondo quanto stabilito nello stato corrente della sessione attiva. Presentiamo i messaggi di handshake nell'ordine in cui devono essere spediti; mandare tali messaggi in ordine diverso provoca sempre un errore fatale.

3.3.6.1 Messaggi di Hello. INIZIO_PAGINA

3.3.6.1 Hello request.

Il messaggio di **hello request** può essere mandato dal server in ogni momento ma sarà ignorato dal client se l'handshake è già stato completato. Esso è una semplice notifica che il client dovrebbe iniziare il processo di negoziazione di nuovo spedendo un messaggio di **client hello.** Dopo aver mandato un **hello request** il server non deve ripeterlo finché la fase di handshake non sarà terminata.

3.3.6.1.1 Client hello.

Quando il client si connette per la prima volta ad un server, deve spedire un messaggio di **client hello** per prima cosa. Il client può anche spedirlo in risposta ad un **hello request**, o per propria iniziativa per rinegoziare i parametri di sicurezza di una connessione esistente. Questo messaggio include una struttura random usata successivamente dal protocollo:

```
struct {
uint32 gmt_unix_time;
opaque random_bytes[28];
} Random;
```

ove `gmt_unix_time` è la data e tempo corrente nel format standard UNIX a 32-bit, e `random_bytes` sono 28 bytes generati da un algoritmo che fornisce numeri casuali.

Il messaggio di **client hello** include anche un identificatore di sessione a lunghezza variabile che, se non vuoto indica una sessione fra il client ed il server i cui parametri di sicurezza sono riusabili dal client; l'identificatore di sessione può provenire da una connessione precedente, dalla connessione attuale o da un'altra correntemente attiva. La struttura usata è la seguente:

```
opaque SessionID<0..32>;
```

La lista `CipherSuite` è anch'essa contenuta in questo messaggio:

```
uint8 CipherSuite[2]; /* selettore di Cryptographic suite */
```

Essa contiene le combinazioni di algoritmi di crittografia supportati dal client, in ordine di preferenza del client stesso. Ogni CipherSuite definisce un algoritmo per lo scambio di chiavi e un CipherSpec. **Il server sceglierà una cipher suite** oppure, se non c'è una scelta accettabile ritornerà un **handshake failure** chiudendo la connessione.

Infine il messaggio client hello contiene **una lista di algoritmi di compressione supportati dal client**, in ordine di preferenza del client: se il server non supporta nessuno di quelli specificati dal client, la sessione deve fallire.

```
enum { null(0), (255) } CompressionMethod;
```

La strutura complessiva del messaggio di **client hello** è quindi la seguente:

```
struct {
ProtocolVersion client_version;
Random random;
SessionID session_id;
CipherSuite cipher_suites<2..215>;
CompressionMethod compression_methods<1..27>;
} ClientHello;

N.B. /* Con le parentesi acute <,> si indica il numero di elementi che un
vettore deve obbligatoriamente contenere: ad esempio nel caso del messaggio
ClientHello, il vettore cipher_suites può contenere da 2 a 215 elementi del tipo
CipherSuite precedentemente definito, ed in ogni caso non può essere vuoto. */
```

Dopo aver mandato un **client hello,** il client aspetta un messaggio di **server hello**, che analizziamo di seguito. Qualsiasi altro messaggio di handshake provocherebbe un errore fatale, tranne un **hello request** che sarebbe ignorato.

Nota implementativa: i dati dell'applicazione non possono essere spediti finché non è spedito un messaggio di **finished**, altrimenti tali dati sarebbero insicuri.

3.3.6.1.2 Server hello. INIZIO_PAGINA

Il server processa il messaggio di **client hello** e risponde con un **server hello** o con un **alert** per il fallimento dell'handshake. La struttura di questo messaggio è simile a quella del **client hello**:

```
struct {
ProtocolVersion server_version;
Random random;
SessionID session_id;
CipherSuite cipher_suite;
```

```
CompressionMethod compression_method;
} ServerHello;
```

ove:

- **`server_version:`** contiene la più bassa versione suggerita dal client e la più alta supportata dal server.
- **`random:`** è una struttura casuale generata dal sever e deve essere diversa e indipendente dal `ClientHello.random`.
- **`session_id:`** è l'identificativo della sessione gestito come illustrato precedentemente.
- **`cipher_suite:`** la singola cipher suite scelta dalla lista fornita dal client in `Client-Hello.cipher_suites`; stabilisce le modalità per l'autenticazione, per cifrare e scambiare le chiavi e i messaggi, come illustrato di seguito nel **paragrafo 3.4.0**.
- **`compression_method:`** è il singolo metodo di compressione scelto dal server dalla lista fornita dal client.

3.3.6.2 Server certificate. INIZIO_PAGINA

Se il server deve essere autenticato, come in genere accade, esso manda il suo certificato immediatamente di seguito al **server hello,** con il messaggio **server certificate.** Il tipo di certificato deve essere adeguato all'algoritmo di scambio di chiavi scelto nella cipher suite, ed è generalmente un X.509.v3. Lo stesso tipo di messaggio è usato per la risposta del client ad un messaggio di **server certificate request.**

3.3.6.3 Messaggio di server key exchange.

Questo messaggio è spedito dal server se esso non ha un certificato, o ha un certificato usato solo per la firma, o se è usato l'algoritmo fortezza/DMS di scambio chiavi.

3.3.6.4 Certificate request.

Un server non anonimo può opzionalmente richiedere un certificato dal client, se ciò è in accordo con la cipher suite concordata.

3.3.6.5 Server hello done. INIZIO_PAGINA

Questo messaggio è mandato dal server per indicare la fine del server hello e dei messaggi associati; dopo aver spedito tale messaggio il server aspetta una risposta dal client. Il quale avendo ricevuto il **server hello done** verifica che il server abbia una valida certificazione, se richiesta, e controlla che i parametri del **server hello** siano accettabili.

3.3.6.6 Client certificate.

Questo è il primo messaggio che il client può mandare dopo aver ricevuto un **server hello done,** se un certificato era richiesto dal server. Se un certificato valido non è disponibile il client può rispondere con un **no certificate alert,** provocando un warning.

3.3.6.7 Messaggio di client key exchange.

La scelta del messaggio di client key exchange è fatta in base a quale algoritmo a chiave pubblica è stato scelto. L'informazione per scegliere la corretta struttura del messaggio è contenuta nel *pending session state.*

3.3.6.8 Messaggio premaster secret cifrato con RSA INIZIO_PAGINA

Se RSA è scelto per lo scambio delle chiavi e l'autenticazione, il client genera un pre-master secret composto da 48 bytes, lo crittografa usando la chiave pubblica del server e spedisce il risultato in un messaggio di **encrypted premaster secret** avente la seguente struttura:

```
struct {
ProtocolVersion client_version;
opaque random[46];
} PreMasterSecret;
```

ove :

- **client_version:** la più nuova versione supportata dal client.
- **random:** 46 bytes generati in modo sicuro.

```
struct {
public-key-encrypted PreMasterSecret pre_master_secret;
} EncryptedPreMasterSecret;
```

in cui :

- **pre_master_secret:** è il valore casuale generato dal client ed usato per generare il master secret come illustrato in seguito.

3.3.6.9 Fortezza key exchange message. INIZIO_PAGINA

Con il Fortezza DMS, il client deriva una Token Encryption Key (TEK) usando Fortezza Key Exchange Algorithm (KEA). In tali calcoli sono usate sia la chiave pubblica del server, ricavata dal suo certificato, sia i suoi parametri privati. Il client genera le chiavi della sessione, le protegge usando il TEK e le spedisce al server, insieme all'IV della sessione; quindi genera un premaster secret di 48 bytes, lo cifra usando il TEK e lo spedisce.

Altri messaggi sono quelli che riguardano la crittografia in modo diretto: **client Diffie-Hellman public value,** analogo al precedente**; certificate verify message** usato per verificare esplicitamente il certificato del client**.** In particolare il messaggio **finished** che è sempre mandato immediatamente dopo un messaggio di **change cipher spec** per verificare che lo scambio e la autenticazione delle chiavi siano andati a buon fine. Questo messaggio di finished è il primo ad essere protetto con le misure di sicurezza appena negoziate; dopodiché le parti possono cominciare a scambiarsi dati confidenziali che saranno così protetti.

3.4 Crittografia. INIZIO_PAGINA

Le modalità per lo scambio delle chiavi, l'autenticazione, la cifratura e gli algoritmi MAC sono determinati dalla **cipher_suite** scelta dal server e rivelata al client nel **server hello message.**

3.4.1 Crittografia asimmetrica o a chiave pubblica.

Gli algoritmi a chiave pubblica sono usati nella fase di handshake per autenticare le parti e generare le chiavi condivise e le altre stringhe casuali.

Per gli algoritmi Diffie-Hellman, RSA, e Fortezza, è usato lo stesso procedimento per convertire la stringa **pre_master_secret** in **master_secret** e cancellare la prima dalla memoria:

```
master_secret =
```

```
MD5(pre_master_secret + SHA('A' + pre_master_secret +

ClientHello.random + ServerHello.random)) +

MD5(pre_master_secret + SHA('BB' + pre_master_secret +

ClientHello.random + ServerHello.random)) +

MD5(pre_master_secret + SHA('CCC' + pre_master_secret +

ClientHello.random + ServerHello.random));
```

3.4.1.1 RSA. INIZIO_PAGINA

Se si usa RSA per l'autenticazione del server e lo scambio delle chiavi, un pre_master_secret di 48 bytes è generato dal client, cifrato con la chiave pubblica del server e spedito. Il server usa la sua chiave privata per metterlo in chiaro, ed entrambi convertono il **pre_master_secret** in **master_secret** come specificato sopra.

3.4.1.2 Diffie-Hellman

E' usato un convenzionale algoritmo Diffie-Hellman; la chiave negoziata è usata come **pre_master_secret** e convertita in **master_secret** nel modo illustrato sopra.

3.4.1.3 Fortezza.

Un **pre_master_secret** di 48 bytes è spedito già cifrato con il TEK. Il server mette in chiaro il pre_master_secret e lo converte in master_secret; in questo caso verrà usato solo per i calcoli del MAC.

3.4.2 Calcoli con la Crittografia Simmetrica e CipherSpec.

La tecnica usata per cifrare e verificare l'integrità dei record SSL è specificata dalla `Cipher Spec` correntemente attiva. Un esempio tipico può essere quello di cifrare i dati usando **DES** e generare i codici di autenticazione usando **MD5.** Per default gli algoritmi di crittografia ed il MAC sono posti a NULL all'inizio: l'Handshake protocol serve appunto per stabilire una `Cipher Spec` sicura e a generare le chiavi.

3.4.2.1 Il master secret. INIZIO_PAGINA

Prima che la cifratura sicura o la verifica dell'integrità possano essere attuate sui records, il client ed il server devono generare dei segreti condivisi noti solo ad essi stessi: questa sequenza di 48 bytes è quella che abbiamo chiamato **master_secret.** Il master_secret è usato per generare le chiavi ed altre stringhe per la crittografia e per i calcoli del MAC.

Le CipherSpecs richiedono un **client write MAC secret, un server write MAC secret, come abbiamo visto nel calcolo del MAC, ed anche un client write key, un server write key, un client write IV, ed un server write IV,** che sono generate a partire dal master_secret e dalle sequenze di bytes specificate in **ClientHello.random** e **ServerHello.random** che agiscono come *entropia* nel processo di generazione. Questo avviene applicando, come mostrato in precedenza, **MD5** sul **master_secret** e sull'**SHA**, calcolato a sua volta sul **master_secret** e sulle due **sequenze random**, concatenando il risultato finché non è raggiunta la dimensione necessaria; a questo punto la sequenza ottenuta è divisa in sottosequenze di dimensione opportuna, che rappresentano i **client write MAC secret, server write MAC secret, client write key, server write key, client write IV, ed il server write IV**, da utilizzare nelle successive operazioni di controllo, autenticazione e cifratura.

3.5 Considerazioni finali e debolezze di SSL. INIZIO_PAGINA

SSL esamina esplicitamente un numero di attacchi, incluso l'attacco forza bruta, crittoanalisi, replay e l'uomo in mezzo. SSL è progettato per agire a livello di rete: ciò significa che non protegge da attacchi agli host, per i quali sarebbe consigliabile proteggersi con un package del tipo Tripwire. I Tripwire usano funzioni hash sicure per assicurare che i documenti non siano cambiati rispetto ad una versione di riferimento protetta in scrittura.

L'uso dell'algoritmo RC4 con chiavi di 40 bits può portare problemi: sembra che un paio di gruppi indipendenti siano già riusciti a forzarlo in circa 8 giorni; sicuramente può quindi essere reso vano da organizzazioni governative o criminali in modo relativamente semplice. Tuttavia la scelta di RC4 è obbligata dalla legge USA sull'esportazione degli algoritmi di crittografia.

Ci sono diversi punti di SSL che, pur non essendo critici, potrebbero offrire ulteriore sicurezza. Ad esempio nel protocollo *SSL Handshake* alcuni dati spediti con il messaggio *CLIENT HELLO* potrebbero essere spediti in un secondo momento, crittografati. Nel protocollo *SSL Record*, un errato *MAC* non dovrebbe far terminare la connessione ma causare una richiesta di ripetizione del messaggio: infatti ci sono pochi attacchi che possono trarre vantaggio dalla duplice spedizione di dati, mentre terminando la connessione si apre la strada agli attacchi basati sul servizio negato; inoltre i numeri di sequenza dovrebbero essere generati in modo casuale invece che ordinatamente. E' importante notare tuttavia che questo approccio alla problematica della sicurezza sembra concepito in modo da permettere facili aggiornamenti ed il supporto a nuovi algortimi di crittografia, senza stravolgere la struttura di SSL.

Fonti su SSL: INIZIO_PAGINA

- Specifiche del protocollo SSLv3.0 in questo Internet Draft : ftp://ietf.cnri.reston.va.us/internet-drafts/draft-freier-ssl-version3-01.txt

- Informazioni da Netscape Communication su:

Secure Sockets Layer (SSL) Protocol: http://www.netscape.com/info/SSL.html

SSL specification(June 1995): http://home.netscape.com/newsref/std/SSL.html

- Informazioni sulla sicurezza:

http://home.netscape.com/newsref/std/index.html ;

http://home.netscape.com/newsref/ref/internet-security.html

Implementazioni di SSL disponibili in rete:

- Implementazione di Netscape Communication di SSLv2.0, SSLref
http://home.netscape.com/info/sslref.html

-Implementazione di SSLv3.01b http://home.netscape.com/newsref/std/sslref.html .

- Versione **SSLLeay 0.6.0** che implementa **SSLv2.0**, per Unix e Windows 3.1/95/NT, Solaris, free sia per uso privato che commerciale, disponibile al: ftp://ftp.psy.uq.oz.au/pub/Crypto/SSL .

- Implementazione di SSLv2.0 di CyberPresence alla pagina http://www.cyberpi.com/, per Windows 3.x/95/NT.

- Secure HTTP e SSL Toolkit commercializzato da Terisa: "http://www.terisa.com/prod/prod.html"

- Versioni dei Web server sicuri di IBM per le maggiori piattaforme: http://www.ics.raleigh.ibm.com/

- Applicazioni basate su SSL:

- **SSLtelnet** Secure telnet basato sul Secure Socket Layer protocol. Disponibile al: ftp.psy.uq.oz.au:/pub/Crypto/SSLapps..
- **SSLftp** Secure ftp basato sul Secure Socket Layer protocol. Disponibile al: ftp.psy.uq.oz.au:/pub/Crypto/SSLapps.
- **Apache-SSL**: Package per adattare il server Apache WWW al supporto SSL. Disponibile al: ftp.hacktic.nl:/pub/replay/pub/apache.

Indirizzi utili:

- W3C, World Wide Web Consortium: http://www.w3.org/hypertext/WWW/Consortium/Prospectus/Overview.html

- Netscape Communications Corp.: http://home.netscape.com

- Per questioni tecniche su SSL e SSLref mandare un email a: ssl-talk@netscape.com .

- EIT; http://www.eit.com (Enterprise Integrated Technologic Inc.).

- IETF; http://www.ietf.org/ .

- Un "secure server " della RSA: https://www.rsa.com/netscape/

- Using RSA Public Key Cryptography : http://www.rsa.com

- SSL - Utili Links: http://www.cs.bris.ac.uk/~bradley/Documents/SSL/links.html

- Utili Links sulla crittografia: ftp.hacktic.nl:/pub/replay/pub/crypto, ftp.funet.fi:/pub/crypt

- Informazioni della Spry C. su Internet Security: http://server.spry.com/secure.htm

- Frequently Asked Question su WWW security: http://www-genome.wi.mit.edu/WWW/faqs/www-security-faq.html

- http://www.psy.uq.oz.au/~ftp/Crypto .

- ftp://ftp.psy.uq.oz.au/pub/Crypto/SSL .

- Damien Doligez's SSL cracking page

- implementazioni e applicazioni SSL :ftp://ftp.psy.uq.oz.au/pub/Crypto/SSLapps .

- http://developer.netscape.com/conference/index.html .

- Hotlist su sicurezza e telematica, diritto e privacy:
http://www.cs.purdue.edu/homes/spaf/hotlists/csec.html

- http://home.netscape.com/eng/mozilla/2.0/handbook/docs/atoz.html#S .

APPENDICE B: RSA

Indice:

Cosa è RSA?

Dopo circa due anni dall'uscita dell'innovativo Diffie-Hellman emerse il primo crittosistema a public-key o asimmetrica, RSA. Oltre ad essere il primo crittosistema di questo genere, rimane ancora oggi il sistema public-key più implementato in assoluto. Prende il nome dai suoi inventori: Ron Rivest, Adi Shamir e Leonard Adleman.
Come già detto, essendo un crittosistema asimmetrico, la chiave con cui viene criptato il messaggio è differente da quella con cui viene decriptato in ricezione. Allora la chiave K sarà formata da Kp, pubblica, e Ks, segreta; le due chiavi possono essere ricavate l'una dall'altra, ma il punto di forza su cui si basa RSA è che l'operazione di derivare la chiave segreta da quella pubblica è troppo complessa per venire eseguita in pratica, anche su un calcolatore molto potente.
In questo tipo di cifrari hanno grande importanza le cosidette funzioni unidirezionali: si tratta di funzioni invertibili tali che il calcolo della funzione diretta sia facile, mentre quello della inversa sia difficile per tutti coloro che non sono in possesso della chiave corretta. Un esempio di tale funzione è il logaritmo in base B dove l'inverso è appunto l'esponenziale. Se la base B è un numero ragionevolmente piccolo, il calcolo sia della funzione diretta che della inversa non risulta particolarmente difficile; diversa sarebbe la situazione se il modulo o base fosse un numero primo enorme, poiché la complessità di calcolo sarebbe a dir poco proibitiva.
La funzione unidirezionale che sta alla base di RSA viene costruita sfruttando il fatto che è facile calcolare il prodotto di due numeri primi molto grandi, ma dato il loro prodotto, è oltremodo difficile risalire ai fattori che lo compongono.
Indichiamo i passaggi del cifrario:

1. Trovare due numeri primi molto grandi P e Q tale che il prodotto o modulo è detto N=PQ;
2. Scegliere E minore di N tale che sia primo con (P-1)(Q-1), il che significa non avere fattori primi in comune. E deve essere dispari. (P-1)(Q-1) non può essere primo perché è pari.
3. Calcolare l'inverso di E, D, tale che DE=1*modulo*(P-1)(Q-1);

N.B. Il *modulo* esegue una operazione di divisione intera tra DE e (P-1)(Q-1) con resto 1.

4. Il testo cifrato si ottiene con l'operazione:

$$C=(T\text{^}E)\textit{modulo}PQ$$

dove T è il plain-text (intero positivo), "^" indica l'esponenziale e C il cipher-text;

5. Il testo decifrato, R, si ottiene così:

$$R=(C\text{^}D)\textit{modulo}PQ$$

dove C indica sempre il cipher-text.

La chiave pubblica è composta da due parti: il modulo N ed E mentre quella privata da N e D, perciò E è definito l'esponente pubblico e D quello privato.
Si può pubblicare la chiave pubblica liberamente perché non si conoscono metodi per calcolare D, P e Q dati N=PQ ed E.
Mostriamo, di seguito, un esempio chiarificatore, assumendo dapprima dei valori piccoli:

- consideriamo P=5 e Q=3, perciò N=15;
- prendiamo E=5, ottenendo quindi D=5;
- il metodo non funziona perché elevando un numero alla quinta potenza ed eseguendo modulo 15, si ottiene come risultato il numero stesso.

Per ottenere un esempio che sia interessante dobbiamo considerare moduli maggiori:

- scelto N=33, avremo Q=11 e P=3;
- prendendo E=3 otteniamo D=7 infatti:
  (P-1)(Q-1)=20 perciò 1*modulo*20=21=DE;
- eleviamo alla terza potenza i numeri da 0-8 ed eseguiamo modulo 33:

plain-text		*cipher-text*
0	_____	0
1	_____	1
2	_____	8
3	_____	27
4	_____	31
5	_____	26
6	_____	18
7	_____	13
8	_____	17

(per esempio: 4^3=64, 64 modulo 33 è uguale a 31)

- per verificare che effettivamente questo cifrario funziona decriptiamo i valori ottenuti al passo precedente: in particolare eleviamo alla settima ed eseguiamo modulo 33.

cipher-text		*plain-text*
0	_____	0
1	_____	1
8	_____	2
27	_____	3
31	_____	4
26	_____	5
18	_____	6
13	_____	7
17	_____	8

(per esempio: consideriamo 13^7=62748517, questo valore modulo 33 mi fornisce il valore 7 che volevamo)

RSA può essere usato sia per autenticare ("authentication") che per motivi di riservatezza ("privacy").

Come scegliere la dimensione della chiave in RSA?

Ricordiamo che la chiave in RSA è costituita da un modulo e un esponente, quindi, quando parliamo di dimensione della chiave, intendiamo quella del modulo N.
La scelta della dimensione di N dipende dal bisogno di sicurezza. Più lungo è il modulo, maggiore è la sicurezza ma anche più lente sono le operazioni di cifratura.
Infatti raddoppiando la lunghezza della chiave si incrementa il numero di operazioni necessarie per la cifratura/decifratura con public-key di un fattore 4, e quelle eseguite con secret-key di un fattore 8. La ragione per cui la cifratura/decifratura con public-key richiede meno operazioni è che l'esponente pubblico E puó rimanere fisso anche se aumenta il modulo. Diversamente l'esponente segreto D deve incrementarsi proporzionalmente ad N.
Si dovrà allora trovare un compromesso determinato su considerazioni riguardanti: l'importanza dei dati da proteggere, e quindi la necessità di sicurezza, ed una valutazione di

quanto potrebbe essere astuto colui che volesse attaccare il cifrario (hacker).
Rivest stima che un modulo di 512 bits può essere fattorizzato con strumenti sofisticati di un valore circa di 8 milioni di dollari, meno nel futuro. Potrebbe perciò essere consigliabile usare un modulo più lungo per es. di 768 bits o ancora maggiore (1024 bits) se i dati da trasmettere sono di estrema importanza.
Generalmente se la chiave è lunga 512 bits ogni fattore primo P e Q deve essere circa 256 bits.

Privacy

Vediamo come utilizzare questo algoritmo per la "privacy" degli utenti di una rete.
Supponiamo che Alice voglia inviare un messaggio M a Bob. Alice crea il testo cifrato C=(M^E)*modulo*N dove E ed N costituiscono la chiave pubblica di Bob. Per decifrare, Bob calcola M=(C^D)*modulo*N con la sua chiave privata <N, D>, ottenendo così il messaggio originale.

Autenticazione

Allo scopo di autenticare un certo messaggio, che verrà trasmesso su una rete di dubbia sicurezza, si può sfruttare tale algoritmo, mostriamo un esempio di tale applicazione.
Mettiamo che Alice voglia inviare un documento firmato a Bob, M. Ella crea una firma digitale S con questa operazione: S=(M^D)*modulo*N dove <N, D> è la sua chiave privata.
Bob riceve tale messaggio e vuole verificare se davvero è stata Alice a scriverlo: quindi calcola (S^E)*modulo*N (con <N, E> chiave pubblica di Alice), se il testo ottenuto è chiaro Bob potrà essere sicuro del mittente.
Ovviamente autenticazione e privacy vengono usate contemporaneamente, cioè una volta verificato che il messaggio è stato inviato da Alice, cosa che tutti potranno fare dal momento che la chiave è pubblica, solo Bob potrà leggere il documento, dato che sarà l'unico a possedere la chiave segreta con cui decifrarlo.

Perché usare RSA rispetto a DES?

RSA non è una alternativa a DES ma un supporto, infatti RSA permette due importanti funzioni a cui non provvede DES:

i. lo scambio sicuro della chiave segreta;

ii. la firma digitale.

Nella maggior parte dei casi vengono usati ambedue questi algoritmi: prima il messaggio è criptato nel modo tradizionale, cioè con DES, tramite una chiave generata casualmente, poi la stessa chiave viene criptata con RSA.
Insieme il cipher-text ottenuto con DES unitamente alla chiave cifrata con RSA è inviato al destinatario.
In ricezione verrà prima decifrata la chiave (crittografia asimmetrica) quindi decriptato il testo con tale chiave nel modo tradizionale (crittografia simmetrica).
In taluni casi RSA non è necessario, per esempio nelle "trasmissioni multiutente", dove le due parti possono mettersi d'accordo sulla chiave durante incontri (meeting) strettamente privati.

In casi che coinvolgono un unico utente, qualora egli volesse conservare un file personale cifrato, non è necessario uno scambio di chiavi.
E' ovvio che se si richiede una firma digitale o una trasmissione di dati tra utenti che non si possono incontrare, sarà necessario usare RSA.
DES è molto più veloce di RSA, a livello di software possiamo dire che è circa 100 volte superiore e, come hardware, dalle 1000 alle 10.000, dipende dalle implementazioni.

Quanto è sicuro RSA?

La rottura di RSA si potrebbe verificare nel caso in cui un attacker scopra la chiave privata corrispondente a quella pubblica; questo permetterebbe agli hacker di leggere tutti i messaggi cifrati con la chiave pubblica e contraffare la firma. Ovviamente per determinare la chiave segreta D l'hacker dovrebbe fattorizzare il modulo N nei due termini P e Q e conoscere l'esponente E.
La fattorizzazione è un'operazione estremamente difficile soprattutto se i numeri coinvolti sono molto grandi.
Per valutare quanto sia laborioso trovare una soluzione a tale problema fu lanciata una sfida, nel 1977, ad un gruppo di volontari che avrebbero dovuto scomporre in fattori primi il numero conosciuto come RSA-129 (di 129 cifre). La sfida rimase priva di risposta fino al 1993 quando Graff, Athins, Leyland e Lenstra decisero di prendere la cosa sul serio.
Furono coinvolte 600 persone e 1600 computer in 25 paesi diversi, che sfruttarono l'implementazione detta "Multiple Polynomial Quadratic Sieve". Dopo circa 8 mesi, il gruppo di volontari riuscì a portare a termine il lavoro, determinando i fattori di RSA-129 oltre 390.000 milioni di anni prima del previsto.
Questo esercizio fatto su RSA-129 dimostra che un modulo di 129 cifre può essere scomposto anche se con forze enormi, per cui data la continua crescita tecnologica dei sistemi di computer e il loro calo nel prezzo bisogna pensare ad un modulo di almeno 1024 cifre per essere sicuri di una protezione a lungo termine.
Un'altra tecnica di rottura di RSA è di determinare M direttamente dall'equazione (M^E)*modulo*N in modo da scoprire il messaggio criptato e contraffare la firma. Nessun attacco di questo tipo è però conosciuto.
Esistono, inoltre, altri metodi che mirano alla scoperta di un singolo messaggio per esempio un semplice messaggio tipo «Attacker at dawn» (attacco all'alba) potrebbe essere decifrato da un hacker anche senza l'aiuto della chiave segreta, oppure, uno stesso testo inviato a più persone ne favorirebbe la decodifica.
Comunque spesso questi attacchi mirano più a rendere insicuro un crittosistema che a violarlo effettivamente.

L'esportazione di RSA

RSA, pubblicato nel 1983, è custodito da PKP (Public Key Patners) e patentato fino al 2000. In Nord America si deve fare richiesta di licenza per usare o vendere RSA, al di fuori invece non ne è concesso l'utilizzo. Ma se un venditore di software, autorizzato all'uso di critttosistemi a public-key, incorpora RSA in un prodotto commerciale, colui che compra il prodotto finito ha diritto legale di usare RSA. Il governo degli Stati Uniti può usare questo algoritmo senza una licenza dal momento che ha partecipato alla sua creazione.
Comunque PKP permette un libero uso, "non commerciale", di RSA con autorizzazioni scritte per ragioni personali, accademiche o intellettuali.
L'esportazione di RSA rispetta la leggi degli Stati Uniti come tutti gli altri prodotti crittografici. Esportare RSA, per scopi di autenticazione, è permesso indipendentemente dalla lunghezza della chiave benché l'esportatore debba dimostrare che il prodotto non può essere facilmente convertito ad un uso di crittazione. Nel caso che RSA venga usato per la privacy il governo USA non permette l'esportazione se la chiave eccede i 512 bits.
RSA è trovato nelle proposte di Internet come standard PEM e PKCS.

CONCLUSIONI

L'esperienza fatta in questo progetto ci è sembrata attuale e molto interessante.

Attuale perché è oggetto di numerosi studi e sta avendo enormi sviluppi: è previsto infatti che a breve termine anche i classici elettrodomestici delle nostre case (frigoriferi, forni, televisori, ecc.) saranno provvisti di Web Server e connessi a Internet, con la possibilità di essere pilotati a distanza.

Interessante perché questo aspetto sta divenendo rilevante anche nel settore industriale dove "l'economia globale" impone la capacità di controllare e manovrare informazioni e dispositivi, in tempo reale, da qualsiasi posizione e in qualsiasi momento.

Le nuove tecniche di automazione industriale sono basate sul principio di intelligenza distribuita dove anche semplici apparecchiature sono dotate di piccoli "computer" con la possibilità di essere gestite e programmate direttamente. Questo risultato è dovuto all'ingresso delle Tecnologie Informatiche in ambito industriale dove una sempre maggiore informatizzazione implica una diminuzione di tempi e distanze, risultando di fondamentale importanza dal punto di vista strategico, logistico e funzionale.

Come detto in precedenza un aspetto fondamentale risulta essere la sicurezza. Per un'industria il rischio di intrusioni e manomissioni deve essere commisurato all'effettivo valore del bene da proteggere. Noi non abbiamo approfondito il problema della sicurezza perché ci eravamo prefissi altri obbiettivi, non quello di implementare un modello di utilizzo nel mondo reale.

RINGRAZIAMENTI

Si ringrazia la ditta ACRAM per la disponibilità, gli strumenti, la documentazione, le informazioni tecniche messi a disposizione e per averci dato l'opportunità di operare in un ambiente lavorativo professionale.