CAN

Controller Area Network

Corso di Reti di Calcolatori

Prof. Orazio Mirabella

Indice generale

1. Overview

Il CAN è un protocollo di comunicazione seriale per applicazioni tempo critiche. È un fieldbus sviluppato intorno alla metà degli anni ottanta dalla compagnia tedesca Robert Bosch per applicazioni automobilistiche ed in seguito usato anche per molte altre applicazioni industriali (sistemi di controllo delle linee produttive, strumentazione nautica, sistemi medici, ...) grazie alle sue caratteristiche di:

- *affidabilità:* eccellenti capacità di individuare e confinare errori per una sicura trasmissione;
- *robustezza:* la rete continua a funzionare anche in seguito al verificarsi di guasti nei nodi o lungo la linea di trasmissione;
- *capacità di funzionamento corretto anche in ambienti difficili* grazie l'uso delle tensioni differenziali nelle linee di comunicazione;
- *espandibilità:* teoricamente è possibile connettere un numero infinito di nodi alla rete. In realtà si hanno vincoli fisici che limitano tale numero;
- *flessibilità:* la configurazione del sistema è molto flessibile grazie al fatto che si fa uso di una comunicazione di tipo **multicast**; nessun nodo ha un indirizzo che lo identifica univocamente all'interno della rete e ciò consente di aggiungere facilmente nuovi nodi senza modificare in alcun modo il software o l'hardware della rete originaria;
- *costi:* le reti CAN presentano bassi costi realizzativi; un nodo CAN costa, infatti, intorno ai 15$ rispetto alle centinaia o migliaia di dollari di altri tipi di nodi (vedi, ad esempio, i nodi ARINC, 1553, ...);
- *transfer rate max di 1-2Mb/s:* il transfer rate è però dipendente dalla lunghezza del bus; si ha un valore di 1Mb/s per bus di lunghezza pari a 40 m circa;
- *access delay max garantito per i messaggi a più alta priorità.*

I principali svantaggi di una rete CAN sono:

- lunghezza massima del bus intorno al km (con valori di bit rate bassi);
- non è garantito un access delay max per i messaggi a più bassa priorità;
- la larghezza di banda non è equamente distribuita tra i vari nodi della rete poiché i meccanismi usati dal CAN sono strettamente legati alle rivalità sul bus le quali vengono risolte per mezzo di identificatori staticamente assegnati ai messaggi della rete.

Le caratteristiche principali del CAN sono:

- è un bus seriale asincrono **multimaster**
- è possibile avere fino a 2032 differenti tipi di messaggi
- un messaggio può contenere da 0 a 8 byte di informazione
- l'accesso al bus è di tipo **CSMA/CD** non distruttivo per il messaggio a più alta priorità
- la comunicazione è di tipo **multicast** e/o **broadcast,** basata su filtri di accettazione dei messaggi

2. Protocollo CAN

2.1 Struttura

L'implementazione proposta del protocollo CAN si basa sul documento ISO TC22/SC3/WG1: *CAN - High Speed - Proposal for an International Draft Standard (15/06/1990)* e segue l'architettura del Modello di Riferimento **OSI** (*Open System Interconnect*) proposto dall'**ISO** (*International Standardization Organization*), stabilendo una rigorosa separazione tra i compiti di ogni strato e consentendo scambi di informazione tra i diversi livelli solo all'interfaccia tra due adiacenti tramite le primitive di servizio

- *request*
- *indication*
- *confirm*

In particolare, dei sette livelli del modello ISO/OSI, il protocollo CAN ne comprende solo i primi due:

- livello 1: *Phisical Layer*
- livello 2: *Data Link Layer*, a sua volta suddiviso nei due sottostrati
  - *logical link control* (LLC)
  - *medium access control* (MAC)

In assenza di un Application layer standard, nelle applicazioni pratiche viene anche incluso un livello utente che si occupa dell'interfacciamento del CAN con le specifiche applicazioni. Va pero' sottolineato come gia' da qualche anno sia in atto una intensa attivita' a livello internazionale per la definizione anche dell'Application layer. Nel frattempo sono gia' disponibili delle versioni proprietarie di Application layer quali:

- xxxxxxxx
- xxxxxxxx.

Il questo tutorial faremo riferimento solo al .....

La componente piu' innovativa del CAN e' il Data Link layer che utilizza un protocollo originale per l'accesso al mezzo fisico. Lo scambio di dati tra i vari utenti e di informazioni di stato tra i vari nodi avviene mediante l'uso di varie categorie di **frame**. Le frame che interessano il livello utente sono:

- *data frame*, usata per inviare dati
- *remote frame*, usata per richiedere dati ad un altro utente

mentre le frame gestite internamente al protocollo e quindi trasparenti all'utente sono:

- *error frame*, usate per segnalare una condizione di errore rilevata da un nodo
- *overload frame*, usata per segnalare le condizioni di sovraccarico di un nodo

2.2 Arbitraggio

Ciascuna frame è caratterizzata da un identificatore che definisce il significato delle informazioni che trasporta.

Ogni bit inviato sul bus da un qualsiasi nodo viene subito sentito su tutto il bus: ciò fa si che tutti i nodi della rete CAN ricevano ogni frame trasmessa. Ogni nodo, conoscendo quali sono gli identificatori dei messaggi di propria competenza, utilizza queste informazioni per filtrare le frame ricevute passando al livello utente solo quelle di propria pertinenza ed ignorando le altre.

Ogni nodo che ha dei dati da trasmettere ascolta il bus e se questo è libero, inizia a trasmettere. Se due o più nodi trasmettono contemporaneamente, si origina una **contesa per l'accesso al bus**: essa viene risolta da un meccanismo di arbitraggio di tipo **CSMA** (*Carrier-Sense Multiple Access*) non distruttivo in grado di garantire la sopravvivenza del messaggio a più alta priorità, senza che avvenga né perdita di tempo né di dati. Questo meccanismo di arbitraggio non distruttivo è reso possibile dalle particolari caratteristiche elettriche del bus fisico, le quali consentono di distinguere due differenti stati (e quindi bit):

- *dominante*, usato per rappresentare il livello logico **0** (***D bit***)
- *recessivo*, usato per rappresentare il livello logico **1** (***R bit***)

Nel caso in cui vengono inviati sul bus contemporaneamente un *D bit* ed un *R bit*, prevale il *D bit*: non avviene alcuna contesa nel senso tradizionale del termine, ma semplicemente il bus assume lo stato elettrico corrispondente al bit dominante (livello logico 0). Possiamo anche interpretare questo comportamento dicendo che il bus effettua istante per istante l'AND logico di tutti i bit inviati dai nodi.

È possibile quindi utilizzare alcuni tra i bit iniziali dei messaggi (***Arbitration Field***) per risolvere la contesa di accesso al bus. Se più nodi tentano di accedere insieme al bus, ognuno di essi invia la propria sequenza di bit di arbitraggio e ascolta contemporaneamente ciò che è presente sul bus (***monitoraggio del bus***): il messaggio più *forte*, ossia contenente più *D bit*, sovrascriverà i messaggi più deboli inviati dagli altri nodi, vincendo quindi la contesa. Il nodo vincitore non si accorge neppure della contesa poiché non rileverà alcun errore di bit durante la sua trasmissione. Tutti gli altri nodi invece, rilevando tali errori, sospenderanno immediatamente la trasmissione ed attenderanno il completamento della trasmissione del nodo vincitore prima di ritentare l'acceso al bus.

Tutti i nodi devono ovviamente iniziare a trasmettere i propri messaggi contemporaneamente, affinché l'eventuale collisione si abbia solo all'interno dell'*Arbitration Field*, inoltre all'interno dell'intera rete ad ogni informazione deve essere assegnato un identificatore univoco.

Condizione indispensabile per il funzionamento del meccanismo di arbitraggio è che tutti i contendenti "vedano" contemporaneamente, lo stesso bit. Ciò corrisponde al fatto che il tempo di propagazione dei bit sia trascurabile rispetto alla durata del bit stesso.

2.3 Stati di funzionamento

Un nodo CAN può essere in uno dei seguenti stati:

- ***Error-Active***
- ***Error-Passive***
- ***Bus-Off***

Lo stato *Error-Active* è lo stato normale di un nodo funzionante correttamente: un nodo *Error-Active* può prendere parte normalmente alle comunicazioni e, nel caso in cui riscontri un errore, trasmette una Active Error Frame. Un nodo che generi o rilevi molti errori (e quindi presumibilmente guasto) modifica il proprio stato in *Error-Passive*: anch'esso può prendere parte alle comunicazioni, tuttavia in caso di errore trasmette una *Passive Error Frame* e quindi attende un intervallo di tempo addizionale prima di poter tornare a trasmettere (si veda il campo *Suspend Transmission* dell'Interframe Space). Se il guasto è di breve durata, il nodo può tornare nello stato *Error-Active* precedente, altrimenti passa nello stato *Bus-Off*. Un nodo *Bus-Off* non prende parte attivamente alle comunicazioni ma è praticamente sconnesso dalla rete CAN. Esso tuttavia continua a monitorare il bus: se il guasto è temporaneo e, dopo un po' di tempo, il nodo non rileva più errori, viene riammesso a pieno titolo nella rete e torna nello stato *Error-Active*.

Lo stato di un nodo è definito dal valore di due contatori di errore e da un insieme di regole che permettono di modificarne i valori. I due contatori di errore sono costituiti da due variabili intere interne ad ogni nodo:

- ***TxCounter*:** contatore degli errori di trasmissione
- ***RxCounter*:** contatore degli errori di ricezione

Il protocollo CAN riesce così a realizzare una strategia di confinamento dei guasti con una tecnica di gestione degli errori in grado di assicurare

- un'affidabile distinzione tra errori temporanei e guasti permanenti
- un'affidabile individuazione ed esclusione automatica dei nodi difettosi

2.4 Physical Layer

Dato che la definizione del *Physical Layer (PL)* non è parte della specifica del CAN, è compito dell'utente scegliere il mezzo trasmissivo più opportuno. La scelta deve tener conto di diversi aspetti come immunità ai disturbi, sovratensioni, possibili rotture del mezzo, radiazioni, potenza assorbita, velocità di trasmissione, lunghezza del cavo e costi. Alcuni mezzi che è possibile usare sono:

- cavo coassiale
- doppino intrecciato (schermato o no)
- fibra ottica

dove il doppino intrecciato è il mezzo di gran lunga più comune.

I segnali sono trasmessi usando tensioni differenziali e i vari moduli CAN della rete sono connessi ad entrambe le linee di segnale (generalmente indicate con 'CAN_H' e 'CAN_L'). Da tale configurazione deriva la caratteristica di fault-tolerance del CAN; infatti, diverse condizioni di guasto (come l'interruzione di una linea di segnale) sono tollerabili dalla rete in quanto consentono ai diversi nodi di continuare a comunicare, sebbene con un più

basso valore del *rapporto segnale/rumore*. L'uso delle tensioni differenziali è utile anche in ambienti estremamente rumorosi in quanto consente di eliminare completamente, o quasi, il rumore anche quando si usa una semplice coppia di fili intrecciati.

I controllori del CAN attualmente disponibili presentano integrate le interfacce del CAN, come il Philips 8051 compatibile col processore 8XC592 e il Siemens SABC167CR. L'80C200 è l'unico controllore CAN che interfaccia direttamente molti microcontrollori. La connessione col mezzo fisico può essere implementata con componenti discreti o con il circuito integrato 82C250. Generalmente l'interfaccia prevede un transceiver on-chip con una configurazione a ponte di transistor per la trasmissione ed un comparatore per la ricezione dei segnali differenziali.

Il bus fisico è realizzato in maniera tale che tutti i nodi 'sentano' contemporaneamente ogni bit presente in esso: ciò garantisce una elevata integrità dei dati, poiché ogni messaggio è contemporaneamente ricevuto da tutti i nodi della rete o da nessun nodo.

La rete CAN inoltre è una rete **multimaster paritetica**, ossia tutti i suoi nodi presentano le stesse opportunità di utilizzo del bus: non appena questo è libero, un qualunque nodo può iniziare a trasmettere un messaggio. Se due o più nodi trasmettono contemporaneamente, si origina una contesa per l'accesso al bus: essa viene risolta da un meccanismo di arbitraggio di tipo **CSMA** (*Carrier-Sense Multiple Access*) non distruttivo in grado di garantire la sopravvivenza del messaggio a più alta priorità, senza che avvenga né perdita di tempo né di dati. Questo meccanismo di arbitraggio non distruttivo è reso possibile grazie alle particolari caratteristiche elettriche del bus fisico, le quali consentono di distinguere due differenti stati:

- ***dominante:*** almeno un trasmettitore invia un livello logico basso
- ***recessivo:*** tutti i trasmettitori inviano un livello logico alto

Da ciò segue la denominazione, nell'ambito del CAN, di *bit dominante* associata al bit 0 (*D bit*) e di *bit recessivo* associata al bit 1 (*R bit*). Come già detto, nel caso in cui vengono inviati sul bus contemporaneamente un *D bit* ed un *R bit*, prevale il *D bit*.

Il *PL* del protocollo CAN utilizza una tecnica tipo ON-OFF per la codifica dei bit denominata *NRZ* (*Non-Return-To-Zero*), secondo la quale ogni bit viene rappresentato mediante un ben preciso livello elettrico: ciò fa sì che una sequenza di più bit uguali generi un unico livello elettrico continuo.

Poiché il clock non è trasportato col segnale, la sincronizzazione è realizzata sul fronte del bit iniziale di ogni **frame**, bit indicato come "Start Of Frame". Allo scopo di mantenere il sincronismo tra i vari nodi durante la trasmissione, le sequenze di bit uguali più lunghe di un limite massimo (generalmente pari a 5) vengono interrotte dall'inserimento forzato di un bit opposto: questo meccanismo è noto come ***bit-stuffing*** e i bit inseriti come *stuff-bit*. Il bit-stuffing è implementato nello strato fisico: esso agisce pertanto in modo trasparente nei confronti di tutti gli strati superiori del protocollo CAN, essendo gli stuff-bit inseriti automaticamente in trasmissione e tolti, sempre automaticamente, in ricezione.

2.5 Data Link Layer

Il DLL è suddiviso nei due sottostrati ***L**ogical **L**ink **C**ontrol* e ***M**edium **A**ccess **C**ontrol*.

2.5.1 LLC

Il sottostrato ***LLC*** (*Logical Link Control*) permette fondamentalmente lo scambio di dati tra una entità di livello utente locale ed una remota in modo affidabile ed efficiente. Per far ciò ogni *LLC* si serve dei servizi messi a disposizione dal proprio sottostrato *MAC*.

Il sottostrato *LLC* fornisce inoltre le seguenti funzionalità:

- ***filtro di accettazione dei messaggi***, basato sul campo *ID* presente nelle *frame LLC*
- ***notifica dei sovraccarichi (overload)***, dovuti a condizioni interne del ricevitore che richiedono un ritardo della successiva *Data* o *Remote Frame*
- ***gestione delle operazioni di recupero***, consistenti nella ritrasmissione automatica delle *frame* corrotte

Lo scambio di informazioni all'*interfaccia livello utente - sottostrato LLC* è mediato da primitive di servizio, *primitive LLC*, indicate anteponendo al nome delle stesse i caratteri *'L_'*.

2.5.1.1 Frame LLC

Le *frame LLC* possono essere di tipo seguente:

Data Frame, utilizzate per lo scambio di dati tra gli utenti di due nodi

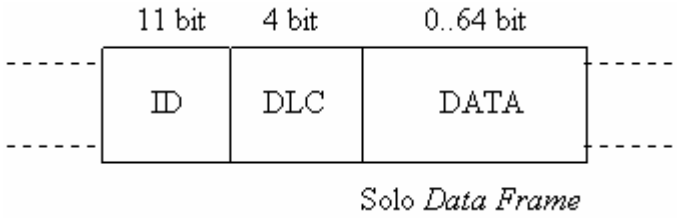
Remote Frame, adoperate dai nodi che desiderano richiedere ad altri nodi la trasmissione di particolari informazioni

Data Frame e Remote Frame LLC

Le *Data Frame* vengono utilizzate per lo scambio di dati tra gli utenti di due nodi, consentendo la trasmissione di un massimo di 8 byte (64 bit).

Le *Remote Frame* sono invece adoperate dai nodi che desiderano richiedere ad altri nodi la trasmissione di particolari informazioni.

Entrambi i tipi di *frame* presentano la struttura mostrata in figura.



Struttura di una *Data* o *Remote Frame LLC*

Il campo ***ID*** (*Identifier*) contraddistingue il tipo di informazione che la *frame* trasporta, ossia il significato dei dati. Il campo ***DLC*** (*Data Length Code*) specifica la lunghezza in byte (tra 0 e 8) del campo *Data* successivo (se *Data Frame*) o dei dati richiesti (se *Remote Frame*). Il campo ***DATA*** contiene, infine, i bit di dati dell'utente da inviare al nodo remoto: ovviamente esso è presente solo nelle *Data Frame* ed assente nelle *Remote Frame*.

Una *Remote Frame* è trasmessa da un nodo per richiedere la trasmissione di una *Data Frame* avente lo stesso campo *ID*.

2.5.1.2 Primitive di Interfaccia Livello Utente - Sottostrato LLC

Le primitive di interfaccia previste dal protocollo CAN tra il livello utente e il *sottostrato LLC* consentono lo scambio di *Data Frame* e di *Remote Frame* tra più entità *LLC*.

Esistono, dunque, due gruppi di primitive:

primitive Data Frame, per la gestione delle *Data Frame*

primitive Remote Frame, per la gestione delle *Remote Frame*

Primitive Data Frame

La tabella seguente descrive brevemente le primitive dedicate alla gestione delle *Data Frame*.

Primitive di interfaccia per la gestione delle *Data Frame*	
L_DATA.Request(*Id,DLC,Data*)	**Significato.** Primitiva passata dal livello utente al *sottostrato LLC* per richiedere l'invio di dati ad una o più entità *LLC* remote. **Parametri.** *Data* contiene le informazioni da inviare. *DLC* ne specifica la lunghezza in byte. *Id* identifica l'informazione e la sua priorità. **Effetto.** L'*LLC* richiede al *MAC* la trasmissione di una *Data Frame LLC*.
L_DATA.Indication(*Id,DLC,Data*)	**Significato.** Primitiva inviata dal *sottostrato LLC* al livello utente per segnalare la ricezione di una *frame* di dati. **Parametri.** *Data* contiene le informazioni ricevute. *DLC* ne specifica la lunghezza in byte. *Id* rappresenta l'identificatore dell'informazione e la sua priorità. **Effetto.** Non specificato.
L_DATA.Confirm(*Status*)	**Significato.** Primitiva inviata dal sottostrato *LLC* al livello utente per segnalare l'esito di una precedente richiesta *L_DATA.Request*. **Parametri.** *Status* può assumere il valore *SUCCESS* o *NO_SUCCESS*. **Effetto.** Non specificato.

Primitive Remote Frame

La tabella seguente descrive brevemente le primitive dedicate alla gestione delle *Remote Frame*.

Primitive di interfaccia per la gestione delle *Remote Frame*	
L_REMOTE.Request(*Id,DLC*)	**Significato.** Primitiva passata dal livello utente al

	sottostrato LLC per richiedere ad una entità *LLC* remota di trasmettere dati. **Parametri.** *DLC* specifica la lunghezza in byte dei dati da inviare. *Id* identifica l'informazione e la sua priorità. **Effetto.** L'*LLC* richiede al MAC la trasmissione di una *Remote Frame LLC*.
L_REMOTE.Indication(*Id,DLC*)	**Significato.** Primitiva inviata dal *sottostrato LLC* al livello utente per segnalare la ricezione di una richiesta remota di trasmissione di dati. **Parametri.** *DLC* specifica la lunghezza in byte. *Id* identifica l'informazione e la sua priorità. **Effetto.** Non specificato.
L_REMOTE.Confirm(*Status*)	**Significato.** Primitiva inviata dal *sottostrato LLC* al livello utente per segnalare l'esito di una precedente richiesta *L_REMOTE.Request*. **Parametri.** *Status* può assumere il valore *SUCCESS* o *NO_SUCCESS*. **Effetto.** Non specificato.

2.5.2 MAC

Il sottostrato ***MAC*** *(Medium Access Control)* permette essenzialmente lo scambio di *frame LLC* tra il sottostrato *LLC* locale e quello remoto; a tale scopo, le *frame LLC* sono mappate nelle *frame MAC*. I servizi forniti dal *MAC* all'*LLC* consentono:

- la trasmissione di dati con **ACK**, di tipo punto-punto, **multicast** o **broadcast** e senza la creazione di una **connessione** DLL
- la richiesta di dati remoti con ACK, anch'essa senza la creazione di una connessione DLL
- la trasmissione di Overload Frame, per ritardare la trasmissione della successiva Data o Remote Frame

Il *MAC* si prende carico dei problemi dovuti alla natura imperfetta del canale fisico di trasmissione e tenta di mascherarne gli errori, fornendo agli strati superiori la visione di un canale praticamente privo di errori.

Il *MAC* fornisce inoltre le seguenti funzionalità :

- incapsulamento e decapsulamento dei dati trasmessi/ricevuti
- rilevazione e segnalazione degli errori
- gestione dell'accesso al mezzo fisico in trasmissione e ricezione

Lo scambio di informazioni all'*interfaccia sottostrato LLC - sottostrato MAC* è mediato da primitive di servizio, *primitive MAC***,** indicate anteponendo al al nome delle stesse i caratteri *'MA_'*.

2.5.2.1 Frame MAC

Le *frame MAC* possono essere di uno qualsiasi dei tipi seguenti:

Data Frame, usate per la trasmissione di Data Frame del sottostrato LLC;

Remote Frame, usate per la trasmissione di Remote Frame del sottostrato LLC;

Error Frame, trasmesse dai nodi CAN Error-Active o Error-Passive (cioè tutti eccetto i nodi Bus-Off) non appena rilevano una condizione di errore in ricezione o trasmissione.

Overload Frame, adoperate per fornire un ulteriore ritardo prima della trasmissione della successiva *Data* o *Remote Frame*.

Interframe Space, usate per separare *Data Frame* e *Remote Frame, ,* o tra loro o da eventuali *Overload Frame* ed *Error Frame* precedenti.

Data Frame e Remote Frame MAC

Le *Data Frame* e le *Remote Frame MAC* sono utilizzate dal sottostrato *MAC* per la trasmissione delle *Data Frame* e *Remote Frame LLC*, che a loro volta trasportano informazioni o richieste di informazioni da parte dell'utente.

Entrambi i tipi di *frame* presentano la struttura mostrata in *Fig. 1*.

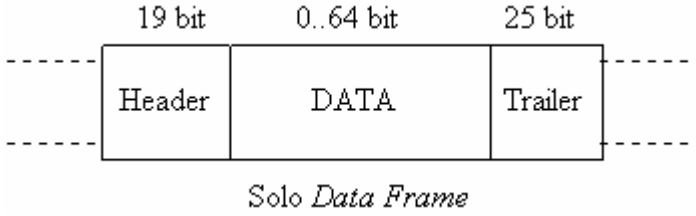


Fig.1*- Struttura di una *Data* o *Remote Frame MAC

Il campo ***DATA*** (presente solo nelle *Data Frame*) è la stessa sequenza di bit contenuta nel campo omonimo delle *Data Frame LLC*; è preceduto da un'intestazione (*header*) e seguito da una coda (*trailer*). I bit dei campi *header* e *trailer* includono informazioni di controllo: essi sono inseriti dal sottostrato *MAC* per l'implementazione effettiva del protocollo tra le due entità *MAC* interessate dalla trasmissione.

L'intestazione (***header***) a sua volta ha la struttura delineata in *Fig.2*

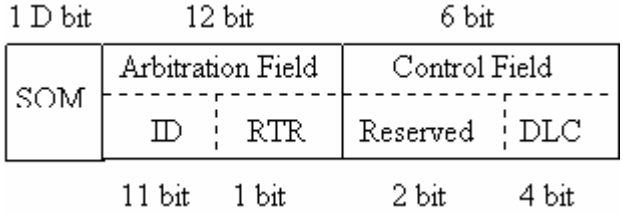


Fig.2 - Struttura dell'intestazione (*header*) di una *Data* o *Remote Frame MAC*

Il bit ***SOM*** (*Start Of Message*) è costituito sempre da un *D bit*: esso marca l'inizio di una *frame MAC* e consente a tutti i nodi di sincronizzarsi sul proprio fronte di salita. Il campo ***ID*** è lo stesso campo presente nelle omonime *frame LLC* ed è adoperato per contraddistinguere il significato delle informazioni trasportate dalla *frame*. Il bit ***RTR*** (*Remote Transmission Request*) serve invece a distinguere una *Data Frame* (se *D bit*) da una *Remote Frame* (se *R bit*). I due campi *ID* e *RTR* sono anche utilizzati per la risoluzione delle eventuali contese di accesso al bus e costituiscono l'***Arbitration Field***. Si

noti che, poiché il bit *RTR* fa parte dell'*Arbitration Field*, le *Data Frame* hanno sempre una priorità maggiore rispetto alle *Remote Frame* a parità di campo *ID*.

Il campo ***DLC*** (lo stesso delle *frame LLC*) specifica infine la lunghezza in byte del campo dati successivo (compresa tra 0 e 8 byte). Il campo *DLC*, insieme ai 2 bit riservati (sempre *D bit* nella presente implementazione) che lo precedono, costituisce il ***Control Field***.

La coda (***trailer***) della *frame* presenta invece la struttura evidenziata in *Fig. 3*.

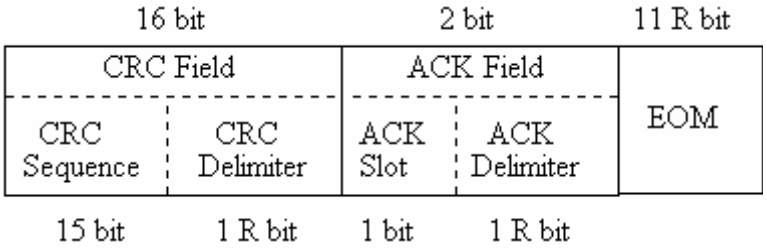


Fig. 3* Struttura della coda (*trailer*) di una *Data* o *Remote Frame MAC

Il campo ***CRC Field*** è costituito dal campo *CRC Sequence* e ***CRC Delimiter*** (sempre un *R bit*). Il campo ***CRC Sequence*** contiene il *CRC* (*Cyclic Redundancy Check*) a 15 bit della *frame* (calcolati su tutti i bit della *frame* a partire dal bit *SOM* e fino all'ultimo bit del campo *Data*): esso è utilizzato per la rilevazione degli errori della *frame*.

Il campo ***ACK Field*** consiste di due bit: l'*ACK Slot* e l'***ACK Delimiter*** (sempre un *R bit*). Il bit ***ACK Slot*** viene riservato per dar modo al nodo destinatario di trasmettere un segnale di corretta ricezione (*Acknowledgement* o *ACK*) al nodo sorgente: esso viene inviato come *R bit* dal nodo sorgente e viene sovrascritto con un *D bit* dal nodo destinatario, nel caso in cui la *frame* sia stata correttamente ricevuta. Se quindi il nodo sorgente non sente un *D bit* in questo bit slot, assume che la frame sia andata persa o sia stata ricevuta errata e può intraprendere le azioni più opportune.

Il campo ***EOM*** (*End Of Message*) contraddistingue infine la fine della *frame* ed è composto da 11 *R bit*.

Si osservi che tutti i bit delle *Data Frame* e *Remote Frame* a partire dal bit di *SOM* fino al campo *CRC Sequence* (incluso) sono interessati dal meccanismo di **bit-stuffing**.

La struttura complessiva di una *Data Frame* o *Remote Frame* è mostrata in *Fig. 4*.

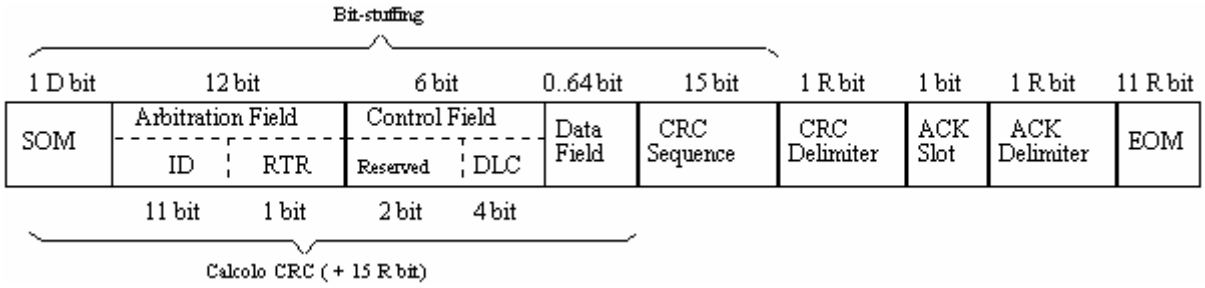


Fig. 4* Struttura di una *Data* o *Remote Frame MAC

Error Frame

Le *Error Frame* vengono trasmesse dai nodi CAN Error-Active o Error-Passive (cioè tutti eccetto i nodi Bus-Off) non appena rilevano una condizione di errore in ricezione o trasmissione. Tuttavia un nodo *Error-Passive* che rileva un errore in ricezione è tenuto ad aspettare 6 bit uguali consecutivi prima di poter trasmettere la propria *Error Frame*.

Esistono due tipi di *Error Frame*:

- ***Active Error Frame*** trasmesse dai nodi *Error-Active*
- ***Passive Error Frame*** trasmesse dai nodi *Error-Passive*

Una *Error Frame* contiene in generale due campi: il campo *Error Flag* e il campo *Error Delimiter*, mostrati in *Fig. 5*. I due tipi di *frame* si differenziano esclusivamente nel campo ***Error Flag***, che nelle *Active Error Frame* è composto da 6 *D bit*, mentre nelle *Passive Error Frame* da 6 *R bit*. Il campo ***Error Delimiter*** è invece composto in entrambe le *frame* da 8 *R bit*.

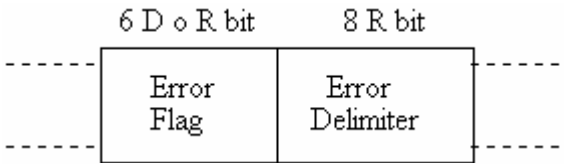


Fig. 5* Struttura di una *Error Frame MAC

Dopo la trasmissione dell'*Error Flag*, il nodo invia una sequenza di *R bit* finché non monitora sul bus un *R bit*. A questo punto invia i rimanenti 7 *R bit* dell'*Error Delimiter*.

Si osservi che sia l'*Active Error Flag* sia il *Passive Error Flag* sono composti da una sequenza di 6 bit identici (*D bit* o *R bit*) e contravvengono perciò alla regola del **bit-stuffing**.

Un *Active Error Flag*, essendo tutto composto da bit dominanti, è in ogni caso rilevato da parte di tutti i nodi della rete poiché vìola la regola del *bit-stuffing* oppure distrugge i campi di bit che richiedono un formato fisso. I nodi della rete reagiscono emettendo a loro volta delle *Error Frame*.

Un *Passive Error Flag* è invece rilevabile dagli altri nodi solo se viene emesso dal nodo trasmettitore della *frame* corrente in un punto in cui provoca una violazione della regola del *bit-stuffing*: è quindi non rilevabile se viene emesso durante l'*Arbitration Field* (poiché può essere sovrascritto da altri bit inviati da altri nodi) oppure nella parte finale di una *Data* o *Remote Frame* (*ACK Delimiter* e *EOM* formano una sequenza di 11 *R bit*).

Overload Frame

Le ***Overload Frame*** sono adoperate per fornire un ulteriore ritardo prima della trasmissione della successiva *Data* o *Remote Frame*. Esse vengono trasmesse dai nodi CAN non appena rilevano:

- una condizione interna di sovraccarico (overload) del ricevitore
- un *D bit* individuato durante l'*Intermission*
- un *D bit* rilevato come ultimo bit di un *EOM*

La trasmissione di una *Overload Frame* può iniziare solo durante l'*Intermission*: in corrispondenza del primo bit di *Intermission* se originata da un sovraccarico interno, oppure al bit successivo dopo il *D bit* rilevato.

Si osservi che il protocollo CAN indica come facoltativo l'invio di una *Overload Frame* in seguito alla ricezione di un *D bit* come settimo bit di un *EOM*.

Un qualsiasi nodo che rileva sul bus una tale *frame* risponde trasmettendo a sua volta una *Overload Frame*, ritardando così la trasmissione della successiva *Data* o *Remote Frame*. Si noti tuttavia che non possono essere generate più di due *Overload Frame* consecutive.

Una *Overload Frame* possiede due campi: il campo ***Overload Flag*** (composto da 6 *D bit*) e il campo ***Overload Delimiter*** (composto da 8 *R bit*), come indicato in *Fig. 6*. Una *Overload Frame* è quindi strutturalmente identica ad una *Active Error Frame*. Tuttavia è distinguibile da essa perché viene generata immediatamente dopo l'ultimo *EOM* della frame precedente e distrugge la struttura dell'*Interframe Space*. Essa è inoltre ricevuta senza che nessun altro nodo abbia rilevato alcun errore. Si osservi che è importante distinguere una *Overload Frame* da una *Active Error Frame* poiché tutti i nodi che la ricevono (anche i nodi *Error-Passive*) devono a loro volta emettere una sequenza di 6 *D bit* consecutivi (*Overload Flag*).

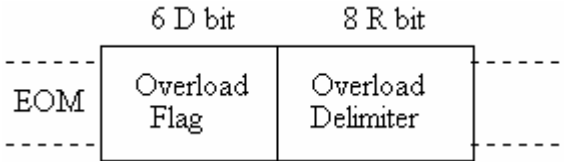


Fig. 6* Struttura di una *Overload Frame MAC

Il ritardo massimo che può essere generato è dunque pari a 43 *bit-time*. Infatti, in seguito alla trasmissione della prima *Overload Frame* si ha un ritardo di 20 *bit-time*:

- 6 *bit-time* dovuti all'*Overload Flag* originale

- 6 *bit-time* dovuti all'*Overload Flag* inviato dagli eventuali nodi che hanno sentito l'*Overload Flag* originale

- 8 *bit-time* necessari a trasmettere l'*Overload Delimiter*

Analogamente, altri 20 *bit-time* di ritardo si hanno a causa della seconda *Overload Frame* trasmessa dal nodo ed ulteriori 3 bit-time di ritardo ritardo sono causati dal campo *Intermission* che è necessario far passare prima di poter trasmettere una *Data* o *Remote Frame.*

Interframe Space

Le *Data Frame* e le *Remote Frame* sono separate tra loro e da eventuali *Overload Frame* ed *Error Frame* precedenti, da una sequenza di bit che prende complessivamente il nome di ***Interframe Space***.

L'*Interframe Space* è composto da **3 *R bit*,** denominati ***Intermission*** e, per i nodi *Error-Passive* che sono stati trasmettitori della *frame* precedente, da altri 8 *R bit*, che prendono il nome di ***Suspend Transmission***. Questi due campi di bit possono essere seguiti dal campo ***Bus Idle***, composto da un numero indefinito di *R bit*, denotanti la disponibilità del bus. Quest'ultimo campo è presente solo se nessun nodo della rete ha alcuna frame da trasmettere.

L'***accesso al bus*** da parte di un nodo può avvenire:

- durante l'*Intermission* per inviare una *Overload Frame*
- subito dopo l'*Intermission* o durante il *Bus Idle* se il nodo è Error-Active oppure anche Error-Passive ma non ha trasmesso la frame precedente
- dopo il campo *Suspend Transmission* se il nodo è *Error-Passive* e ha trasmesso la frame precedente
- in un qualsiasi istante nel caso in cui venga individuato un errore

Il campo *Suspend Transmission* costituisce dunque una penalizzazione addizionale per i nodi *Error-Passive*. I bit di *Bus Idle* indicano invece che il bus è libero e può quindi essere utilizzato da un qualsiasi nodo per trasmettere una *Data Frame* o una *Remote Frame*.

In condizioni di funzionamento normale pertanto il bus contiene le sequenze di bit mostrate in *Fig. 7* le quali si ripetono indefinitamente (o almeno finché esistono nodi che si scambiano dati o richieste).

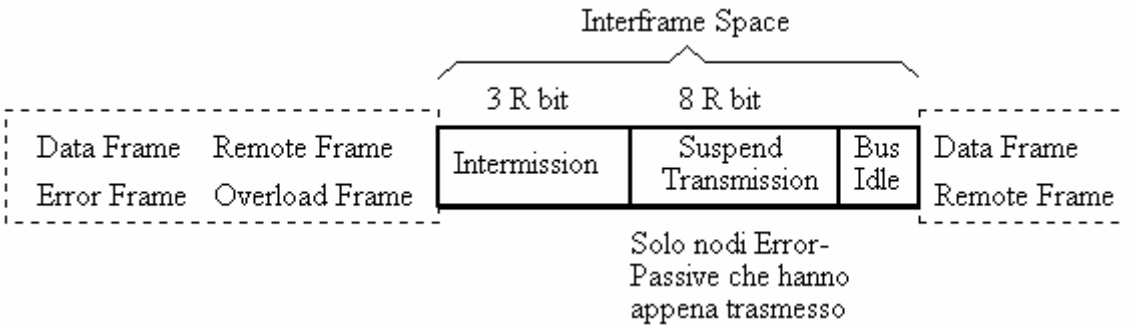


Fig.7 Struttura dell'*Interframe Space*

2.5.2.2 Primitive di Interfaccia Sottostrato LLC - Sottostrato MAC

Le primitive di interfaccia tra il *sottostrato LLC* ed il *sottostrato MAC*, previste dal protocollo CAN, permettono, oltre allo scambio di *Data Frame* e di *Remote Frame*, anche l'invio di *Overload Frame*.

Esistono dunque tre gruppi di primitive:

primitive Data Frame, permettono la trasmissione di *Data Frame MAC*

primitive Remote Frame, permettono la trasmissione di *Remote Frame MAC*

primitive Overload Frame, permettono l'invio di *Overload Frame MAC*

Primitive Data Frame MAC

La tabella seguente descrive brevemente le primitive dedicate alla gestione delle *Data Frame*.

Primitive di interfaccia per la gestione delle *Data Frame*	
MA_DATA.Request(*Id,DLC,Data*)	**Significato.** Primitiva passata dal *sottostrato LLC* al *sottostrato MAC* per richiedere l'invio di dati ad una o più entità *MAC* remote. **Parametri.** *Data* contiene le informazioni da inviare. *DLC* ne specifica la lunghezza in byte. *Id* identifica l'informazione e la sua priorità. **Effetto.** Il *MAC* prepara la propria *Data Frame* e quindi la invia al *Physical Layer* per la trasmissione effettiva.
MA_DATA.Indication(*Id,DLC,Data*)	**Significato.** Primitiva inviata dal *sottostrato MAC* al *sottostrato LLC* per segnalare la ricezione di una *frame* di dati. **Parametri.** *Data* contiene le informazioni ricevute. *DLC* ne specifica la lunghezza in byte. *Id* identifica l'informazione e la sua priorità. **Effetto.** Non specificato.
MA_DATA.Confirm(*Status*)	**Significato.** Primitiva inviata dal *sottostrato MAC* al *sottostrato LLC* per segnalare l'esito di una precedente richiesta *MA_DATA.Request*. **Parametri.** *Status* può assumere il valore *SUCCESS* o *NO_SUCCESS*. **Effetto.** Non specificato.

Primitive Remote Frame MAC

La tabella illustra sinteticamente le primitive dirette alla gestione delle *Remote Frame*.

Primitive di interfaccia per la gestione delle *Remote Frame*	
MA_REMOTE.Request(*Id,DLC*)	**Significato.** Primitiva passata dal *sottostrato LLC* al *sottostrato MAC* per richiedere ad una entità *MAC* remota di trasmettere dati. **Parametri.** *DLC* specifica la lunghezza in byte dei dati da inviare. *Id* identifica l'informazione e la sua priorità. **Effetto.** Il *MAC* prepara la propria *Remote Frame* e quindi la invia al *Physical Layer* per l'effettiva trasmissione.
MA_REMOTE.Indication(*Id,DLC*)	**Significato.** Primitiva inviata dal *sottostrato MAC* al *sottostrato LLC* per segnalare la ricezione di una richiesta remota di trasmissione di dati. **Parametri.** *DLC* specifica la lunghezza in byte. *Id* identifica l'informazione e la sua priorità. **Effetto.** Non specificato.
MA_REMOTE.Confirm(*Status*)	**Significato.** Primitiva inviata dal *sottostrato MAC* al *sottostrato LLC* per segnalare l'esito di una precedente richiesta *MA_REMOTE.Request*. **Parametri.** *Status* può assumere il valore *SUCCESS* o *NO_SUCCESS*. **Effetto.** Non specificato.

Primitive Overload Frame MAC

La tabella che segue mostra sommariamente le primitive rivolte alla gestione delle *Overload Frame*.

Primitive di interfaccia per la gestione delle *Overload Frame*	
MA_OVLD.Request()	**Significato.** Primitiva passata dal *sottostrato LLC* al *sottostrato MAC* per richiedere la trasmissione di una *Overload Frame*. **Effetto.** Il *MAC* costruisce una *Overload Frame* e quindi la invia al *Physical Layer* per la trasmissione.
MA_OVLD.Indication()	**Significato.** Primitiva inviata dal *sottostrato MAC* al *sottostrato LLC* per segnalare la ricezione di una *Overload Frame*. **Effetto.** Non specificato.
MA_OVLD.Confirm(*Status*)	**Significato.** Primitiva inviata dal *sottostrato MAC* al *sottostrato LLC* per segnalare l'esito di una precedente richiesta *MA_OVLD.Request*.

	Parametri. *Status* può assumere il valore *SUCCESS* o *NO_SUCCESS*. **Effetto.** Non specificato.

2.5.3 Flusso delle primitive

Il flusso delle primitive durante la trasmissione di dati tra due generici nodi j e k della rete ha inizio con l'invio da parte del nodo j, al sottostrato *LLC* locale, della richiesta di trasmissione dei dati mediante la primitiva *L_Request*. L'entità *LLC* passa a sua volta la richiesta al *MAC* mediante la primitiva *MA_Request* e ne segnala l'esito all'utente mediante la primitiva *L_Confirm*.

Il *MAC* costruisce un messaggio secondo il proprio protocollo, invia i dati al *MAC* remoto, sfruttando i servizi dello strato fisico e quindi segnala all'*LLC* l'esito della precedente *L_Request* locale per mezzo della primitiva *L_Confirm.*

Il sottostrato *MAC* del nodo k, ricevuti i dati dallo strato fisico, segnala l'evento all'*LLC*, utilizzando la primitiva *MA_Indication*. A sua volta l'*LLC* invia la segnalazione appropriata al livello utente mediante la primitiva *L_Indication*.

2.5.4 Gestione degli Errori

2.5.4.1 Overview

È compito del *sottostrato MAC* occuparsi della rilevazione, segnalazione e gestione degli errori nelle *frame*, allo scopo di fornire agli strati superiori la visione di un canale praticamente privo di errori. Tale obiettivo è raggiunto mediante l'utilizzo di due contatori di errore, costituiti da due variabili intere *TxCounter* (contatore degli errori di trasmissione) ed *RxCounter* (contatore degli errori di ricezione) per ciascun nodo CAN, e di un insieme di regole che permettono di modificarne i valori.

A seconda dei valori che questi contatori assumono, il nodo può essere in ogni istante in uno dei seguenti tre stati di funzionamento:

- *Error-Active* se entrambi i contatori sono minori di 128
- *Error-Passive* se uno dei due contatori è maggiore o uguale di 128
- *Bus-Off* se *TxCounter* è maggiore o uguale di 256

Inizialmente i due contatori di errore *TxCounter* e *RxCounter* sono posti a 0 e il nodo si trova nello stato *Error-Active*; ciò denota un nodo affidabile (avente cioè sia il trasmettitore sia il ricevitore funzionanti correttamente) che partecipa a pieno titolo allo scambio di informazioni. Se un nodo *Error-Active* riscontra un errore, trasmette delle *Active Error Frame* e incrementa *TxCounter* se l'errore è rilevato in trasmissione, *RxCounter* se l'errore è rilevato in ricezione. Se non riscontra alcun errore, il relativo contatore viene decrementato.

Il tasso di incremento dei contatori (normalmente 8) è generalmente maggiore di quello di decremento (solitamente 1), il che corrisponde ad assegnare un peso maggiore agli errori rispetto ai successi. Possiamo anche dire che dopo un errore un nodo deve comportarsi correttamente per un periodo di tempo relativamente lungo prima che l'errore venga *dimenticato.*

Quando *TxCounter* oppure *RxCounter* oltrepassano la soglia di 127, il nodo modifica il proprio stato in *Error-Passive*, continua a prendere parte alla comunicazione ed invia *Passive Error Frame* se rileva errori. Appena raggiunge lo stato *Error-Passive*, un nodo risulta penalizzato in trasmissione dovendo attendere un intervallo di tempo addizionale (il campo *Suspend Transmission* dell'*Interframe Space* - 8 R bit) prima di poter tornare a ritrasmettere.

Se il contatore *TxCounter* raggiunge od oltrepassa il valore 256, il nodo entra nello stato *Bus-Off* e risulta praticamente sconnesso dalla rete CAN. Esso tuttavia continua a monitorare il bus; se riceve 128 sequenze di 11 *R bit* (ad esempio *ACK Delimiter* + *EOM* + *Interframe Space* oppure *Error* o *Overload Delimiter* + *Interframe Space*), quindi praticamente dopo 128 *frame* corrette, viene riammesso a pieno titolo nella rete e ritorna nello stato *Error-Active* azzerando *TxCounter* e *RxCounter*.

Un nodo *Error-Active* viene considerato funzionante, per cui l'effetto delle *Active Error Frame* equivale ad una cancellazione delle *frame* errate con la conseguente ritrasmissione delle stesse. Un nodo *Error-Passive* è invece probabilmente malfunzionante per cui ad esso viene dato minor credito da parte degli altri nodi della rete e penalizzata la sua partecipazione alla comunicazione.

Osserviamo che un trasmettitore *Error-Active*, affetto da un guasto *grave* di cui è in grado di accorgersi, si porta il più rapidamente possibile (compatibilmente con l'incremento fissato per i contatori di errore) nello stato *Error-Passive*, bloccando in questa fase qualunque altro scambio di messaggi sul bus (a causa dell'invio delle proprie *Active Error Frame*). Infatti non appena rileva un errore, incrementa il contatore relativo ed inizia a trasmettere una *Active Error Frame*. Se il guasto permane, il nodo rileverà altri errori anche durante la trasmissione dell'*Error Frame*, i quali provocheranno ulteriori incrementi dei contatori e l'invio di altre *Error Frame.* In questo modo, se i contatori sono inizialmente 0, è sufficiente la rilevazione di 16 errori consecutivi per cambiare lo stato in *Error-Passive*. Una volta nello stato *Error-Passive*, l'invio di ulteriori *Error Frame* (*Passive Error Frame* per la precisione) non è in grado di bloccare il bus.

2.5.4.2 Rilevazione degli Errori

Il *sottostrato MAC* fornisce i seguenti meccanismi di rilevazione degli errori in una *frame*:

Monitoraggio del bus. È effettuato continuamente dal nodo trasmettitore dopo ogni bit inviato sul bus; se viene rilevata una differenza tra questo bit e quello monitorato viene generato un **bit error**. Ovviamente non sono considerati errori di bit quelli rilevati durante la fase di arbitraggio e nell'*ACK Slot*.

Controllo della regola del *bit-stuffing.* È effettuato dai nodi ricevitori sui campi interessati dal *bit-stuffing*, ossia dal bit di *SOM* fino all'ultimo bit del campo *CRC Sequence* di una *Data* o *Remote Frame*; la violazione della regola, ossia la rilevazione di una sequenza di 6 bit uguali, provoca un **stuff error**.

Controllo del formato della *frame.* È svolto dai nodi ricevitori sui campi delle *Data* o *Remote Frame* caratterizzati da un formato fisso: *CRC Delimiter*, *ACK Delimiter* e *EOM*

(ad eccezione dell'ultimo bit che, se ricevuto come *D bit*, viene considerato una segnalazione di *overload*). Se il formato atteso non è rispettato viene generato un **form error**.

Controllo del *CRC.* I nodi ricevitori calcolano il *CRC* a 15 bit della *frame* ricevuta, utilizzando lo stesso algoritmo adoperato dal nodo trasmettitore; se esiste una discordanza tra il *CRC* così calcolato ed il campo *CRC Sequence* ricevuto, viene generato un **CRC error**.

Controllo dell'*ACK.* Il nodo trasmettitore emette un *R bit* durante l'*ACK Slot* e si aspetta di monitorare un *D bit* inviato dal nodo destinatario in caso di corretta ricezione; se ciò non accade, viene generato un **acknowledgement error**.

Osservazioni:

- Un nodo *Error-Passive* ignora eventuali errori di bit durante la trasmissione della *Passive Error Frame*: infatti questa può essere sovrascritta da altri nodi *Error-Active* che hanno rilevato lo stesso errore.
- Un nodo ricevitore non considera errore di bit la ricezione di un *D bit* come ultimo bit dell'*EOM* di una *frame*, ma può inviare facoltativamente una *Overload Frame*.

2.5.4.3 Segnalazione degli Errori

Non appena viene rilevato un qualsiasi errore, il *MAC* informa il *sottostrato LLC* e trasmette una *Error Frame.* Se si tratta di un **CRC error**, il *MAC* attende l'*ACK Delimiter* e quindi inizia a trasmettere l'*Error Frame*, a meno che nel frattempo non abbia rilevato una ulteriore condizione di errore che a sua volta ha causato l'invio di una *Error Frame*. Un qualsiasi altro tipo di errore (bit, *stuffing*, form, *ACK*) provoca invece la trasmissione immediata (in corrispondenza del bit successivo) di una *Error Frame* da parte del *MAC*. L'*Error Frame* è inviata sia dal nodo trasmettitore (sia *Error-Active* che *Error-Passive*), se si accorge dell'errore, sia da tutti i nodi ricevitori *Error-Active* che rilevano l'errore. Si osservi che i nodi ricevitori *Error-Passive* sono esclusi dalla segnalazione; non avrebbe infatti senso una segnalazione da parte loro, perché le *Passive Error Flag* (composte da 6 *R bit*) verrebbero sicuramente sovrascritte dal nodo trasmettitore in quel momento attivo (se *Error-Active* e ha rilevato l'errore oppure anche se è *Error-Passive* ma non si è accorto dell'errore) o da eventuali altri nodi ricevitori *Error-Active*.

2.5.4.4 Regole di modifica dei contatori di errore

I valori dei due contatori di errore di un nodo vengono incrementati se il nodo rileva un errore di trasmissione o di ricezione. L'ammontare dell'incremento dipende dal tipo di errore rilevato, dalla posizione dell'errore nella *frame* ed infine dallo stato del nodo. I contatori vengono invece decrementati nel caso in cui il nodo trasmetta o riceva una *frame* correttamente.

Il protocollo definisce due insiemi di regole di modifica dei contatori di errori:

regole di modifica di TxCounter

regole di modifica di RxCounter

Esaminando le regole fissate dal protocollo, sembra ragionevole ipotizzare che la filosofia seguita dal protocollo CAN nei casi di errore dubbi (in cui può trattarsi sia di un errore del trasmettitore che del ricevitore) sia di ipotizzare che il ricevitore del nodo funzioni correttamente.

2.5.4.4.1 Regole di modifica di *TxCounter*

Il protocollo CAN stabilisce la seguente serie di regole per la modifica dei valori del contatore di errori di trasmissione *TxCounter:*

1. Nodo trasmettitore *Error-Active* che rileva un (monitora un *R bit* invece di un *D bit*) durante la trasmissione di una *Active Error Flag*:

- incrementa *TxCounter* di 8
- invia una *Active Error Frame*

È plausibile che l'errore sia dovuto ad un guasto del ricevitore del nodo.

2. Nodo trasmettitore *Error-Passive* che rileva un (un *D bit* invece di un *R bit*) durante la trasmissione di una *Passive Error Flag*:

- non incrementa *TxCounter*
- non invia alcuna nuova *Passive Error Frame*

Questo evento non è considerato un errore dal nodo; probabilmente ciò si può spiegare ipotizzando che il *D bit* monitorato sia stato inviato da un altro nodo *Error-Active* che abbia rilevato lo stesso errore che ha causato l'invio della *Passive Error Frame*. Non ha senso inviare in questo caso una nuova segnalazione di errore, dato che il nodo ha comunque già incrementato il proprio *TxCounter*. Osserviamo che questa regola costituisce un'ulteriore conferma del fatto che un nodo *Error-Passive* non è considerato molto affidabile, neppure quando rileva degli errori.

3. Nodo trasmettitore *Error-Passive* che rileva un ACK error (non individua alcun *D bit* di *ACK*) nell'*ACK Slot* e non rileva alcun *D bit* durante la trasmissione del *Passive Error Flag*:

- non incrementa *TxCounter*
- invia una *Passive Error Frame*

In questo caso infatti il nodo ha trasmesso correttamente la *frame*, la quale semplicemente non è stata riconosciuta; non si tratta quindi di un errore proprio, ma piuttosto di un errore dei nodi ricevitori. Questa regola serve anche ad assicurare che nella rete rimanga sempre almeno un nodo in grado di trasmettere, evitando che tutti i nodi della rete diventino *Bus-Off*.

4. Nodo trasmettitore che rileva uno stuff error durante la fase di arbitraggio dovuto ad uno *stuff-bit* (prima del bit di *RTR*) che è stato inviato correttamente come recessivo ma monitorato come dominante:

- non incrementa *TxCounter*
- invia una *Error Frame*

Questa regola risulta giustificata considerando che l'errore è dovuto probabilmente ad un *Active Error Flag* o ad un *Overload Flag* inviato da un altro nodo.

5. Nodo trasmettitore che rileva un gruppo di 8 *D bit* consecutivi dopo un *Error Flag* o che rileva ogni ulteriore gruppo di 8 *D bit* consecutivi:

- incrementa *TxCounter* di 8
- non invia alcuna *Error Frame*

Con molta probabilità questi bit sono causati da errori permanenti nel nodo o nella linea.

6. Nodo trasmettitore non contemplato nei casi precedenti che rileva una condizione di errore :

- incrementa *TxCounter* di 8
- invia una *Error Frame*

7. Nodo trasmettitore che trasmette una *Data Frame* o una *Remote Frame* senza errori fino all'ultimo bit del campo *EOM* e riceve l'*ACK* di conferma nell'*ACK Slot*:

- decrementa TxCounter di 1, se TxCounter > 0

2.5.4.4.2 Regole di modifica di RxCounter

Il protocollo CAN stabilisce la seguente serie di regole per la modifica dei valori del contatore di errori di ricezione *RxCounter:*

1. Nodo ricevitore che rileva un bit error in una *Data* o *Remote Frame*:

- incrementa *RxCounter* di 1
- trasmette una *Error Frame*

2. Nodo ricevitore *Error-Active* che rileva un bit error durante la trasmissione di una *Active Error Flag*:

- incrementa *RxCounter* di 8
- invia una *Active Error Frame*

In questo caso l'errore è dovuto con molta probabilità ad un guasto del ricevitore (monitora un *R bit* dopo aver inviato sul bus un *D bit*).

3. Nodo ricevitore *Error-Passive* che rileva un bit error durante la trasmissione di una *Passive Error Flag* (monitora un *D bit* invece di un *R bit*):

- non incrementa *RxCounter*
- non invia alcuna nuova *Passive Error Frame*

L'errore viene in effetti ignorato; questo comportamento è giustificato dal fatto che con ogni probabilità il *D bit* monitorato è stato inviato da un altro nodo *Error-Active* che ha individuato lo stesso errore che ha provocato l'invio della *Passive Error Frame* da parte del nodo. Questo errore di bit non fa altro che confermare al nodo che si tratta di un errore altrui e l'invio del *Passive Error Frame* sarebbe in questo caso ridondante.

4. Nodo ricevitore che rileva un bit error nella trasmissione di una *Overload Flag* (in risposta ad una *Overload Frame* precedentemente monitorata):

- incrementa *RxCounter* di 8
- invia una *Error Frame*

Anche in questo caso, come nella regola 1, l'errore è dovuto probabilmente ad un guasto del ricevitore (un *D bit* è monitorato come *R bit*).

5. Nodo ricevitore che rileva un bit error (monitora un *D bit* invece di un *R bit*) come primo bit susseguente la trasmissione di un *Error Flag* (ossia mentre tenta di inviare l'*Error Delimiter* e quindi è in attesa di ascoltare un *R bit*):

- incrementa *RxCounter* di 8
- non invia alcuna *Error Frame*

6. Nodo ricevitore che rileva un gruppo di 8 *D bit* consecutivi dopo un *Error Flag* o che rileva ogni ulteriore gruppo di 8 *D bit* consecutivi:

- incrementa *RxCounter* di 8
- non invia alcuna *Error Frame*

Probabilmente questi bit sono dovuti ad errori permanenti del nodo o della linea.

7. Nodo ricevitore che riceve una *Data Frame* o una *Remote Frame* senza errori fino all'*ACK Slot* e che invia l'*ACK* senza errori:

- decrementa *RxCounter* di 1, se *RxCounter 127; se invece RxCounter* > 127, pone *RxCounter* pari ad un valore compreso tra 119 e 127

Osserviamo che un nodo Error-Passive torna nuovamente nello stato Error-Active alla prima ricezione di una frame priva di errori (se TxCounter < 128).

3
Smart Distribuited System

Specifica protocollo Application Layer

3.1 Introduzione

L'SDS (*Smart Distributed System*) è stato ideato per essere di supporto in applicazioni di automazione industriale, sebbene il suo uso sia stato esteso in altri campi. Tipicamente può essere usato in quelle applicazioni che richiedono dispositivi come sensori fotoelettrici, sensori di prossimità, limit switches, valvole elettro-pneumatiche, relè, controllori, pannelli di interfaccia per operatore, servo motori ecc.

L'SDS è stato progettato dalla Honeywell per soddisfare esigenze di *velocità* (può essere usato per sistemi tempo critici), *affidabilità* (grazie alle capacità di rilevazione e correzione degli errori e allo scambio di messaggi con acknowledge), *flessibilità* (possibilità di espandere i modelli dei dispositivi per non fare diventare obsoleto il sistema) tipiche di applicazioni di produzione automatica.
Il protocollo dell'SDS è implementato facendo riferimento ad una forma compatta del l'**ISO/OSI Reference Model** costituita da tre dei sette strati previsti dallo standard: *Phisical Layer, Data Link Layer, Application Layer*.

La specifica dell'Application Layer SDS è stata ottimizata per l'uso con il Data Link Layer del protocollo CAN (BOSCH V2.0A CAN specification); comunque, grazie all'architettura a strati cui fa riferimento, il protocollo SDS può essere utilizzato con diversi tipi di protocolli Data Link Layer.

3.2 Modello dell'SDS

Una rete SDS consiste di vari dispositivi fisici collegati tra loro tramite un mezzo fisico.

I vari dispositivi vengono modellati come degli oggetti (**Object Models**) e di questi vengono specificati gli attributi, gli eventi e le azioni che supportano:

- *attributi*: ciascuna specifica di attributo include AttributeId, Name, tipo di dato rappresentato (Read Only oppure Read/Write), descrizione.
- *eventi*: sono messaggi non sollecitati, generati dal modello del dispositivo; ciascun evento è descritto da EventId, Name, Type of Data.
- *azioni*: ciascuna specifica di azione include ActionId, Name, Input/Output Parameter, Type Parameter, descrizione.

Per garantire l'interoperabilità tra i dispositivi, modellati come oggetti, l'SDS prevede una struttura di tipo gerarchica. In alto alla gerarchia si trova il *Minimum Model* che specifica un insieme minimo di caratteristiche e funzionalità (eventualmente ereditate a loro volta da altri modelli standard che precedono il Minimum Model nella gerarchia) che ogni dispositivo SDS deve supportare per assicurare l'interoperabilità. Sotto il Minimum Model sono definite 4 categorie di modelli:

I. dispositivi di I/O

II. funzioni IEC 1131-3
III. blocchi funzioni SDS
IV. interfacce SDS

Per utilizzare un dispositivo, o si adotta un modello standard definito in una delle 4 categorie, oppure si specifica un nuovo modello che eredita da una delle 4 categorie di modelli le funzionalità di base e per il quale si definiscono attributi, azioni ed eventi caratteristici del dispositivo che si sta definendo.

3.3 Servizi dell'Application Layer SDS

I servizi messi a disposizione dall'Application Layer SDS sono stati pensati per offrire il massimo supporto nelle reti SDS realizzate con il CAN.

I servizi disponibili sono:

Read: permette di leggere il valore dell'attributo di un dispositivo da parte dell'ALP-service-user.

Write: permette di modificare il valore dell'attributo di un dispositivo da parte dell'ALP-service-user.

Event Report: permette di notificare un evento da parte dell'ALP-service-user.

Action: permette all'ALP-service-user di comandare un dispositivo affinchè esegua un'azione.

Change of State ON: permette di notificare il cambio di stato a ON di un modello.

Change of State OFF: permette di notificare il cambio di stato a OFFdi un modello.

Write ON State: permette di settare a ON lo stato di un dispositivo di I/O.

Write OFF State: permette di settare a OFF lo stato di un dispositivo di I/O.

L'SDS Service Model usa le quattro primitive *request*, *response*, *indication*, *confirm* per fornire i servizi dell'Application Layer al Data Link Layer del CAN.

3.3.1 Read

Il servizio Read è usato per leggere il valore di un attributo di un dispositivo SDS (ad es.: leggere il valore corrente di un sensore). La seguente tabella descrive i parametri definiti nella primitiva di servizio Read:

PARAMETER	Request	Indication	Response	Confirm	FUNCTION
Address	Mandatory	Come Request	Mandatory	Come Response	...
Logical Device Address	Mandatory	Come Request	Mandatory	Come Response	Definisce il dispositivo dal quale l'attributo deve essere letto
Embedded Device Id	Mandatory	Come Request	Mandatory	Come Response	Specifica quale embedded device deve essere letto.
Attribute Id	Mandatory	Come	Mandator	Come	Specifica l'attributo

		Request	y	Response	che deve essere letto. Il valore del parametro è definito nel modello del dispositivo dell'embedded device.
Result (+) **Attribute Id** **Attribute Value**	...	...	Selective Mandatory Mandatory	Come Response	Se la lettura ha avuto successo. Essa ritorna Attribute Id e il valore che è stato letto.
Result (-) **Attribute Id** **Error Code**	...	...	Selective Mandatory Mandatory	Come Response	Se la lettura è fallita. Essa ritorna Attribute Id e l'Error Code che specifica il motivo del fallimento.

Quando una Read Service è invocata, il **sender** trasmette una **Read APDU Request** al Logical Device Address. Il **receiver** riceve una **Read Indication**. Se la lettura fallisce, il receiver emette una primitiva Response contenente il parametro Result(-) e una **Read APDU Response** è mandata indietro al sender. Se la lettura ha successo, la primitiva Response contiene il parametro Result(+) e una **Read APDU Response** contenente il dato richiesto è mandata indietro al sender. Il sender riceve la primitiva **Confirm** indicante il parametro Result(+) o Result(-) dell'APDU corrispondente.

3.3.2 Write

Il servizio Write è usato per modificare il valore di un attributo di un dispositivo SDS (ad es.:settare l'uscita di un attuatore a ON o OFF). La seguente tabella descrive i parametri definiti in questa primitiva di servizio:

PARAMETER	Request	Indication	Response	Confirm	FUNCTION
Address	Mandatory	Come Request	Mandatory	Come Response	...
Logical Device Address	Mandatory	Come Request	Mandatory	Come Response	Definisce il dispositivo che contiene il target Device
Embedded Device Id	Mandatory	Come Request	Mandatory	Come Response	Specifica quale embedded device deve essere letto
Attribute Id	Mandatory	Come Request	Mandatory	Come Response	Specifica l'attributo che deve essere scritto. Il valore del parametro è definito nel

					modello del dispositivo dell'embedded device
Attribute Value	Mandatory	Come Request	...	...	Valore al quale l'attributo deve essere settato
Result (+) **Attribute Id**	...	...	Selective Mandatory	Come Response	Se la scrittura ha avuto successo. Essa ritorna Attribute Id.
Result (-) **Attribute Id** **Error Code**	...	...	Selective Mandatory Mandatory	Come Response	Se la scrittura è fallita. Essa ritorna Attribute Id e l'Error Code che specifica il motivo del fallimento.

Quando un Write Service è invocato, il **sender** trasmette una primitiva **Write APDU** al **receiver**. Il receiver riceve una **Write Indication**. Se la scrittura fallisce, il receiver emette una primitiva Response contenente il parametro Result(-) invece, se la lettura ha successo la primitiva Response contiene il parametro Result(+); in entrambi i casi, una **Write APDU Response** è mandata indietro al sender. Il sender riceve la primitiva **Confirm** indicante il parametro Result(+) o Result(-) dell'APDU corrispondente.

3.3.3 Event Report

Il servizio Event Report è usato per riportare l'occorrenza di eventi definiti nella specifica del modello di un dispositivo SDS (ad es.: un dispositivo potrebbe effettuare un 'self test failure'). La seguente tabella descrive i parametri definiti in questa primitiva di servizio:

PARAMETER	Request	Indication	Response	Confirm	FUNCTION
Address	Mandatory	Come Request	Mandatory	Come Response	...
Logical Device Address	Mandatory	Come Request	Mandatory	Come Response	Definisce il dispositivo in cui si è verificato l'evento.
Embedded Device Id	Mandatory	Come Request	Mandatory	Come Response	Specifica quale embedded device sta trasmettendo.
Event Id	Mandatory	Come Request	Mandatory	Come Response	Specifica l'evento che si è verificato. Il valore del parametro è definito nel modello del dispositivo dell'embedded device.
Event	Conditiona	Come	...	...	Dipendono dal

Parameters	I	Request			tipo di evento e sono definiti dalle specifiche del modello del dispositivo.
Result (+) **Event Id**	...	...	Selective Mandatory	Come Response	Se l'Event Report ha avuto successo. Ritorna Event Id.
Result (-) **Event Id** **Error Code**	...	...	Selective Mandatory Mandatory	Come Response	Se l'Event Report fallisce. Essa ritorna Event Id e l'Error Code che specifica il motivo del fallimento.

Quando un Event Service è invocato, il **sender** trasmette una primitiva **Event APDU** al **receiver**. Il receiver riceve una **Event Indication** dal Data Link Layer e legge l'APDU. Il receiver genera, poi, una Event Response APDU riportando un successo ed emette una primitiva Response. Il sender riceve la primitiva Confirm con parametro Result(+). Se il receiver non è capace di riportare l'evento, il sender riceve una primitiva con parametro Result(-).

3.3.4 Action

Il servizio Action è usato per eseguire le azioni specificate per un dispositivo SDS (ad es.: per forzare una variabile d'uscita). La seguente tabella descrive i parametri definiti in questa primitiva di servizio:

PARAMETER	Request	Indication	Response	Confirm	FUNCTION
Address	Mandatory	Come Request	Mandatory	Come Response	...
Logical Device Address	Mandatory	Come Request	Mandatory	Come Response	Definisce il dispositivo che contiene il target device.
Embedded Device Id	Mandatory	Come Request	Mandatory	Come Response	Specifica quale embedded device deve eseguire l'azione.
Action Id	Mandatory	Come Request	Mandatory	Come Response	Specifica l'azione che deve essere eseguita. Il valore del parametro è definito nel modello del dispositivo del target device.
Action Parameters	Conditional	Come Request	...	...	Dipendono dal tipo di azione e sono definiti dalle

					specifiche del modello del dispositivo.
Result (+) **Action Id** **Action Results**	...	...	Selective Mandatory Conditional	Come Response	Se la richiesta di azione ha avuto successo. Ritorna Action Id e Action Results.
Result (-) **Action Id** **Error Code**	...	...	Selective Mandatory Mandatory	Come Response	Se la richiesta di azione fallisce. Essa ritorna Action Id e l'Error Code che specifica il motivo del fallimento.

Quando un Action Service è invocato, il **sender** trasmette un **Action APDU** al **receiver**. Il receiver riceve una **Action Indication** dal Data Link Layer e legge l'APDU. Se il receiver non riesce a localizzare l'azione che deve essere eseguita, l'Action Service fallisce e il receiver genera una APDU con parametro Result(-). Se l'Action Service fallisce per un cambio di indirizzo, non viene generata alcuna risposta. Se l'Action Service ha successo, il receiver genera una APDU con parametro Result(+).

3.3.5 Change of State ON

Il servizio Change of State ON è usato da un dispositivo SDS di I/O per riportare l'occorrenza di un cambiamento del suo stato in ON. Questo servizio è fornito per aumentare l'efficienza della rete quando i dispositivi di I/O supportano un unico dispositivo Embedded I/O (se sono supportati più dispositivi Embedded I/O, il cambiamento di stato in ON è riportato con un Event Service). Le caratteristiche fisiche dello stato ON sono definite dalle specifiche del modello del dispositivo di I/O. La seguente tabella descrive i parametri definiti in questa primitiva di servizio:

PARAMETER	Request	Indication	Response	Confirm	FUNCTION
Address	Mandatory	Come Request	Mandatory	Come Response	...
Logical Device Address	Mandatory	Come Request	Mandatory	Come Response	Definisce il dispositivo di I/O in cui si è verificato il cambiamento di stato in ON.
Result (+)	...	...	Selective	Come Response	Se il servizio si è concluso con successo.

Quando un Change of State ON Service è invocato, il dispositivo di I/O genera una **Change of State ON APDU** ed emette una primitiva di Request. Il receiver riceve una **Change of State ON Indication** e la **Change of State ON APDU.** Il Data Link Layer genera una Change of State ON ACK APDU ed emette una primitiva di Response. Il sender riceve la primitiva Confirm e l'APDU con parametro Result(+). Se si verifica un errore, nessuna risposta viene tornata.

3.3.6 Change of State OFF

Il servizio Change of State OFF è usato da un dispositivo SDS di I/O per riportare l'occorrenza di un cambiamento del suo stato in OFF. Questo servizio è fornito per aumentare l'efficienza della rete quando i dispositivi di I/O supportano un unico dispositivo Embedded I/O (se sono supportati più dispositivi Embedded I/O, il cambiamento di stato in OFF è riportato con un Event Service). Le caratteristiche fisiche dello stato OFF sono definite dalle specifiche del modello del dispositivo di I/O. La seguente tabella descrive i parametri definiti in questa primitiva di servizio:

PARAMETER	Request	Indication	Response	Confirm	FUNCTION
Address	Mandatory	Come Request	Mandatory	Come Response	...
Logical Device Address	Mandatory	Come Request	Mandatory	Come Response	Definisce il dispositivo di I/O in cui si è verificato il cambiamento di stato in OFF.
Result (+)	...	...	Selective	Come Response	Se il servizio si è concluso con successo.

Quando un Change of State OFF Service è invocato, il sender genera una **Change of State OFF APDU** che trasmette al receiver. Il receiver riceve una **Change of State OFF.** Il Data Link Layer genera una Change of State ON ACK APDU ed emette una primitiva di Response. Il sender riceve la primitiva Confirm e l'APDU con parametro Result(+). Se si verifica un errore, nessuna risposta viene tornata.

3.3.7 Write ON State

Il servizio Write ON State è un servizio di scrittura specializzato, usato da un dispositivo SDS di I/O per settare a ON lo stato di un dispositivo di I/O. Questo servizio è fornito per aumentare l'efficienza della rete quando i dispositivi di I/O supportano un unico dispositivo Embedded I/O (se sono supportati più dispositivi Embedded I/O, lo stato del dispositivo deve essere settato a ON con il servizio Write). Le caratteristiche fisiche dello stato ON sono definite dalle specifiche del modello del dispositivo di I/O. La seguente tabella descrive i parametri definiti in questa primitiva di servizio:

PARAMETER	Request	Indication	Response	Confirm	FUNCTION
Address	Mandatory	Come Request	Mandatory	Come Response	...
Logical Device Address	Mandatory	Come Request	Mandatory	Come Response	Definisce il dispositivo di I/O che deve assumere lo stato ON.
Result (+)	...	...	Selective	Come Response	Se il servizio si è concluso con successo.

Quando un sender genera una **Write ON State APDU** e la trasmette a destinazione, il receiver riceve una **Write ON State Indication.** Il Data Link Layer usa la primitiva Response per generare una Write ON ACK APDU. Il sender riceve la primitiva Confirm e l'APDU con parametro Result(+). Se si verifica un errore, nessuna risposta viene tornata.

3.3.8 Write OFF State

Il servizio Write OFF State è un servizio di scrittura specializzato, usato da un dispositivo SDS di I/O per settare a OFF lo stato di un dispositivo di I/O. Questo servizio è fornito per aumentare l'efficienza della rete quando i dispositivi di I/O supportano un unico dispositivo Embedded I/O (se sono supportati più dispositivi Embedded I/O, lo stato del dispositivo deve essere settato a OFF con il servizio Write). Le caratteristiche fisiche dello stato OFF sono definite dalle specifiche del modello del dispositivo di I/O. La seguente tabella descrive i parametri definiti in questa primitiva di servizio:

PARAMETER	Request	Indication	Response	Confirm	FUNCTION
Address	Mandatory	Come Request	Mandatory	Come Response	...
Logical Device Address	Mandatory	Come Request	Mandatory	Come Response	Definisce il dispositivo di I/O che assume lo stato OFF.
Result (+)	...	...	Selective	Come Response	Se il servizio si è concluso con successo.

Quando un sender genera una **Write OFF State APDU** e la trasmette a destinazione, il receiver riceve una **Write OFF State Indication.** Il Data Link Layer genera una Write OFF ACK APDU ed emette una primitiva Response. Il sender riceve la primitiva Confirm e l'APDU con parametro Result(+). Se si verifica un errore, nessuna risposta viene tornata.

3.4 Protocollo dell'Application Layer SDS

I servizi dell'SDS sono realizzati mediante l'uso di **APDU**; alcuni campi delle APDU sono codificati in un formato fisso, mentre altri campi contengono un numero variabile di parametri (si tratta generalmente di dati o di tipo predefinito nello standard SDS - Boolean, Unsigned Integer, Signed Integer, Real Number, Character String, Date - oppure di un tipo costruito a partire da quelli predefiniti). Le APDU definite nell'SDS possono essere di 2 tipi:

- **Short Form APDU**: è usata per settare lo stato di un dispositivo che supporta un unico embedded device con un solo input o un solo output (single point device);
- **Long Form APDU** : è usata per i messaggi SDS che richiedono più informazioni di quanto la Short Form APDU non possa fornire. Essa è utile per leggere attributi, riportare informazioni di diagnostica, ..., e per settare lo stato di un dispositivo che supporta più embedded device.

3.4.1 Short Form APDU

Si compone dei seguenti campi:

- *Direction Bit* [1 bit]: indica la direzione della comunicazione.
- *Device Address* [7 bit]: contiene l'indirizzo del partner coinvolto nella comunicazione. Se è *Direction Bit*=1 l'indirizzo si riferisce al dispositivo sorgente; se è *Direction Bit*=0 l'indirizzo si riferisce al dispositivo destinazione.
- *APDU Type* [3 bit]: indica uno dei seguenti 8 servizi Change of State OFF/ON/OFF ACK/ON ACK, Write OFF/ON State, Write OFF/ON State ACK.
- *Remote Transmission Request* [1 bit]: è sempre 0.
- *Data Length Code* [4 bit]: è sempre 0.

3.4.2 Long Form APDU

Si compone dei seguenti campi:

- *Direction Bit* [1 bit]: indica la direzione della comunicazione.
- *Device Address* [7 bit]: contiene l'indirizzo del partner coinvolto nella comunicazione. Se è *Direction Bit*=1 l'indirizzo si riferisce al dispositivo sorgente; se è *Direction Bit*=0 l'indirizzo si riferisce al dispositivo destinazione.
- *Long APDU Type* [3 bit]: indica uno dei seguenti 4 servizi Write, Read, Action, Event (le altre codifiche sono riservate).
- *Remote Transmission Request* [1 bit]: è sempre 0.
- *Data Length Code* [4 bit]: indica il numero di byte di cui si compone il dato. E' sempre maggiore di 0.
- *APDU Type Modifier* [8 bit]: questo campo si compone a sua volta in:
  A. *APDU Type* [2 bit]: indica se il messaggio è una Request, una Response con successo, una Response con insuccesso (Error).
  B. *Fragmentation Bit* [1 bit]: indica se l'APDU è stata frammentata in quanto troppo lunga.
  C. *Embedded Object Id* [5 bit]: è l'identificativo dell'embedded device cui l'APDU si riferisce.
- *Variable Id* [8 bit]: se il campo *Long APDU Type* indica un servizio Write o Read, *Variable Id* specifica l'attributo che deve essere letto o modificato, se indica un servizio Action o Event, *Variable Id* specifica il tipo di azione desiderata o l'evento che deve essere riportato.
- *Fragment Number*: numero di sequenza del frammento di messaggio.
- *Total Number of Variable Data Bytes*: numero totale di byte ricevuti attraverso tutti i frammenti.
- *Variable Data*: se il campo *Long APDU Type* indica un servizio Write o Read, *Variable Data* specifica il valore della variabile che è stata letta o modificata, se indica un servizio Action o Event, *Variable Data* fornisce i parametri supportati dal tipo specifico di azione o evento.

3.5 Considerazioni

È doveroso a questo punto sottolineare alcune caratteristiche del protocollo:

***Indirizzamento**:* ogni componente fisico connesso al bus di una rete SDS consiste di uno o più dispositivi logici, ciascuno con un proprio indirizzo SDS (CAN). Ogni dispositivo logico è a sua volta composto da uno o più **Embedded I/O Device** o altri **Object Models** fino ad un massimo di 32 entità (il campo Embedded Object Id della **Long Form APDU** è lungo 5 bit).

***Priorità**:* il primo byte di ogni APDU SDS (sia Short Form che Long Form) è composto dai campi *Direction Bit* e *Device Address*. Il byte, nella sua interezza, definisce la priorità del messaggio: un valore più basso indica una priorità più alta.

***Frammentazione**:* se la lunghezza di un messaggio supera la lunghezza massima delle PDU del CAN, è necessaria la frammentazione della APDU. L'APDU viene suddivisa in "frammenti" numerati in sequenza e tutti con stesso **header**; solo i campi **Fragment Number** e **Variable Data** cambiano. È compito del receiver riassemblare l'intero messaggio a partire dal contenuto del campo *Fragment Number* di ciascun frammento. I messaggi frammentati sono intesi, tipicamente, per messaggi non tempo critici. Pertanto, per permettere l'accesso al bus da parte di altri dispositivi, si lascia trascorrere un tempo pari a 10 ms tra l'emissione di due frammenti successivi. La frammentazione non è necessaria per le Short Form APDU.