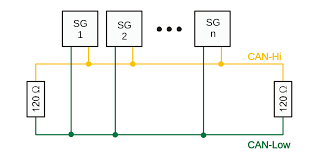
# Documentazione

Immaginiamo di avere una rete CAN, con un certo numero di centraline:



Quello che vogliamo realizzare è un livello applicativo, che ci permette di rappresentare in modo standard i nostri dati, questo è utile poiché il nostro dato dovrà essere preso e portato all’esterno, tuttavia dobbiamo avere a disposizione alcune informazioni, che la frame CAN così com’è fatta, non è in grado di fornire. Nel dettaglio le informazioni che ci interessano riguardano la centralina che ha prodotto quella specifica frame, la centralina che deve ricevere quella specifica informazione, il contenuto informativo trasportato cosa questo contenuto rappresenta e tra gli 8 byte dei dati dove andare a trovare lo specifico contenuto informativo. Per fare questo, consideriamo una frame CAN con 11 bit-identifier, il cosiddetto standard 2.0B.

La struttura della frame CAN 2.0B è questa:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S  O  M | 29 bit Identifier | R  T  R | R0 | DLC | 0…8 Bytes Data | CRC | ACK | E  O  M |

Dei 29 bit di identificativo consideriamo questa rappresentazione:

|  |  |  |  |
| --- | --- | --- | --- |
| 3 bit Priority | 10 bit Reserved | 8 bit Destination Address | 8 bit Source Address |

In questo modo possiamo considerare che il campo Priority ci indica queste informazioni:

* I primi 3 bit ci indicano la priorità durante l’arbitration
* Fornisce 8 livelli di priorità
* Il valore di (000) = priorità maggiore
* Il valore di (111) = priorità minore

Per il source Address possiamo considerare queste caratteristiche:

* Il Source Address = rappresenta gli ultimi 8 bit del message identifier;
* Il Source Address identifica in maniera univoca l’ECU che ha trasmesso un messaggio;
* Ogni indirizzo deve essere univoco all’interno della rete.

Per il Destination Address possiamo considerare queste caratteristiche:

* Destination Address = i successivi 8 bit al campo reserved del message identifier;
* Il Destination Address identifica in maniera univoca l’ECU che deve ricevere il messaggio;

I 10 bit Reserved rappresentano dei bit che verranno usati per sviluppi futuri.

Lo step successivo è conoscere cosa rappresenta il contenuto informativo che si trova all’interno del payload, per fare questo consideriamo che questa informazione ci viene data dall’ID stesso della frame. Come sappiamo il nostro payload è caratterizzato da 8 byte, consideriamo che di questi 8 byte, uno specifico contenuto informativo, può essere presente solo in alcuni di questi byte trasportati. Questo può essere fatto considerando una sorta di database che ci permette di capire dall’ID a cosa si riferisce l’informazione, quindi in maniera standard io posso prelevare una specifica informazione andando a prendere il byte corrispondente, facciamo un esempio:

|  |
| --- |
| Parameter Group: Engine Temperature  Number = 162  Engine collant temperature |
| Byte 1  Fuel Temperature |
| Byte 2 |
| Byte 3,4  Engine Oil Temperature |
| Byte 5,6  ……. |
| Byte 7 |
| Byte 8 |

Una volta capita l’informazione, questa è estratta e convertita in un valore ad alto livello, cioè il nome della centralina univoco, il tipo d’informazione (ad esempio Fuel Temperature,…) , e il contenuto informativo, quindi il valore. A questo punto, è creato un messaggio CoAP e all’interno del payload, che è rappresentato come un array di byte dove sono inserite le informazioni estratte dalla frame CAN in ogni byte, quindi questa informazione può essere inviata a un server CoAP che gestisce tali informazioni.

Nel caso di MQTT è fatta la stessa cosa, l’unica differenza è che avremmo un publisher che trasmette tale informazione verso il broker.

# Implementazione programma

Per quanto riguarda il programma java implementato, andiamo a considerare le classi che sono rappresentate e quali sono i metodi implementati da tali classi.

CAN\_FRAME: la classe in questione, ci rappresenta la struttura della frame CAN, in realtà in questo sistema rappresentiamo solo i campi che ci interessano per costruire lo strato applicativo. Gli unici metodi presenti sono i metodi di get e set.

CENTRALINA: questa classe contiene le informazioni che riguardano l’identificativo univoco della centralina, il nome della centralina che rappresenta il nome del dato più importante prodotto dalla centralina e tutti gli id prodotti da quella centralina. I metodi presenti, sono i metodi di get e set, quindi abbiamo un metodo insert che è utilizzato per inserire in una lista i diversi indirizzi prodotti dalla centralina.

INFORMAZIONE: la classe informazione, contiene come attributi le informazioni che dobbiamo estrarre dalla frame CAN, nello specifico troviamo come attributi:

* Nome: rappresenta il dato che andiamo a prelevare dalla frame (es: velocità, Fuel Temperature,…);
* Num\_Byte\_Payload: questo indica nel nostro payload quale byte dobbiamo andare a controllare nel payload stesso, che conterrà il dato rappresentato.
* Id\_Centralina: identifica univocamente la centralina che ha prodotto l’informazione;
* Indirizzo: rappresenta l’Identificativo univoco della frame CAN, che ci permette di estrarre tutte queste informazioni.
* Id\_Centralina\_Dest: identifica univocamente la centralina di destinazione.

DATABASE: questa classe è utilizzata per andare a costruire il nostro database nel veicolo, cioè per capire dato l’ID, quale centralina ha prodotto la frame in questione, quale centralina riceverà l’informazione, il dato specifico trasportato, il byte in cui andare a trovare una specifica informazione.

Quello che andiamo a fare in questa classe è andare a costruire una lista, dove ogni elemento della lista contiene un oggetto informazione, la cui classe abbiamo descritto in precedenza.

COSTRUZIONE\_FRAME: questa classe è utilizzata per simulare il caso in cui è ricevuta una frame dal livello 2 e sono prelevate da questa frame le informazioni corrispondenti per andare a costruire un oggetto frame che sarà poi usato per estrarre l’informazione in questione.

ESTRAZIONE: questa classe è utilizzata per prendere le informazioni che ci interessano nella frame che dovrà poi essere trasmessa all’esterno. Nello specifico gli attributi che andiamo a rappresentare sono:

* Id\_centralina: identifica univocamente la centralina che ha prodotto quell’informazione;
* Nome: rappresenta il dato che andiamo a prelevare dalla frame (es: velocità, Fuel Temperature,…);
* Dato: il campo payload dove troviamo il contenuto informativo.
* Nome\_Centralina: l’informazione più importante trasportata.

CONVERTER: questa classe è utilizzata per estrarre l’informazione dalla frame, per poi rendere disponibile questa informazione a uno dei protocolli di livello applicativo, che hanno il compito di trasmettere all’esterno l’informazione.

Per fare questo utilizziamo il costruttore, che per prima cosa, chiama il metodo constuct\_Frame utilizzato per assegnare i diversi campi della frame, quindi è inizializzato il database, questo è importante perché ci consente di allocare l’oggetto che contiene tutte le informazioni inserite nel database.

A questo punto è creato il metodo information\_data (); questo metodo ci permette di ottenere l’id della frame che viaggia sulla rete CAN, come abbiamo detto in precedenza, l’id ci serve per accedere alle informazioni che ci interessano per rappresentare la frame. Per prima cosa, andiamo a vedere se l’id di destinazione coincide con la centralina, a questo punto vado a vedere se l’id è presente nel database, non appena riesco a trovare l’indice, lo step successivo è quello di andare a estrarre il byte del payload che contiene una specifica informazione; poi è memorizzato il contenuto informativo e inserito in una variabile. Andiamo a inserire dentro una lista di oggetti di tipo estrazione tutte le informazioni estratte dalla frame CAN.

A questo punto prendiamo questa lista d’informazione estratte e le inviamo all’esterno utilizzando sia il protocollo MQTT sia CoAP, questo è fatto all’interno della classe Client.

Vediamo il diagramma UML del sistema:

