Design Pattern #2 – Simple State Machine

Category: LabVIEW Talk
scritto da Davide Oratelli | Marzo 17, 2020



Simple State Machine Pattern (SSM)

Nel precedente articolo abbiamo visto il primo modello di progettazione, il General VI Pattern, oggi incominceremo a parlare di macchina a stati.

Ogni programmatore è alla ricerca di un modo più rapido e semplice per sviluppare software, uno strumento adatto è la macchina a stati.

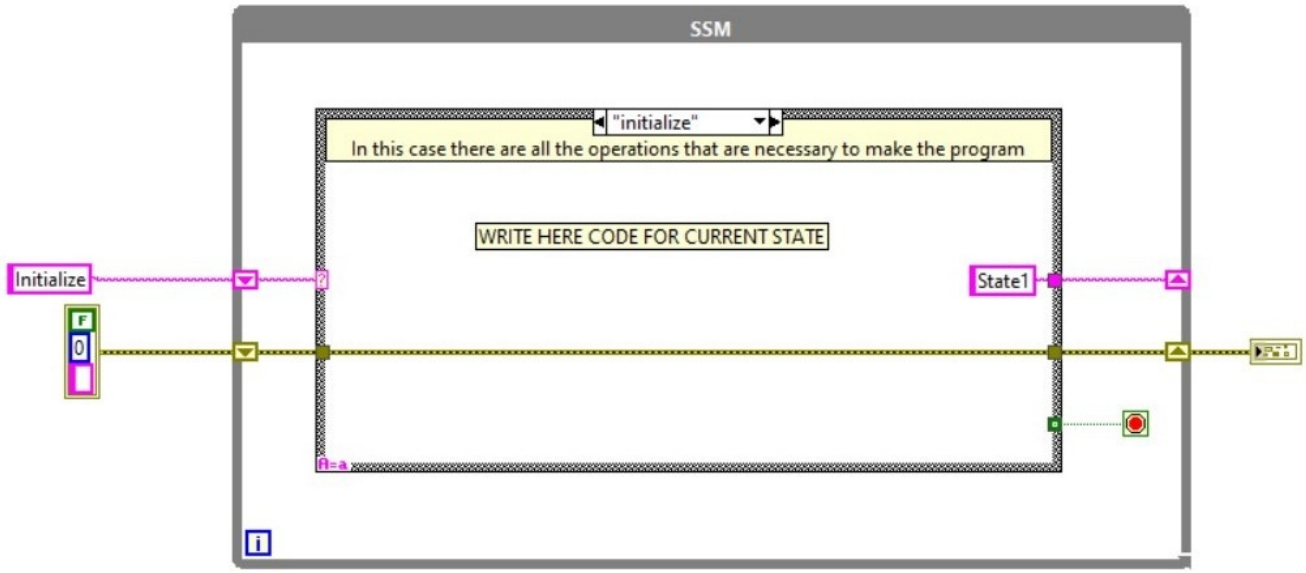
Ma cos'è la macchina a stati in labVIEW?

Detta in parole semplici la Simple State Machine Pattern consiste in una Case Structure all' interno di un While Loop, e almeno uno shift register per memorizzare lo stato corrente all'iterazione.

Il ciclo while offre la possibilità di eseguire il codice fino a quando al conditional terminal viene dato valore falso (es. un bottone).

Il Case Structure consente di variare il codice da eseguire, all' interno di una macchina a stati semplice vi sono diversi casi, ma solitamente sono già determinati nell' iterazione precedente.

La Simple State machine viene usata per VI che possono essere facilmente divisibili per compiere multiple azioni divise per casi. Questo Pattern è composto da una fase Startup, una fase Shutdown e differenti codici/azioni eseguiti dalla fase principale dell' applicazione, in base a condizioni che si verificano nel codice.



Bytelabs- SSM

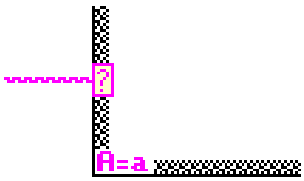
Solitamente all' interno del codice è presente una costante enum che deve essere definita(TypeDef), gli enumeratori sono molto utili quando si parla di state machine perché rendono il tutto più leggibile e i casi possibili saranno solo quelli esplicitati nella typedef.

Questo vantaggio però limita la flessibilità e l'utilizzo di funzionalità offerte invece utilizzando come messaggio di transizione, le stringhe.

L'utilizzo di stringhe ha il contro che è potenzialmente più facile scrivere messaggi che non trovano corrispondenza di caso, ma utilizzando degli accorgimenti questo unico svantaggio è eliminato.

Il primo accorgimento è utilizzare lo stato di Default che verrà eseguito ogni volta non ci sia un match tra il messaggio e il case. Questo per noi sarà un caso da gestire come errore di sviluppatore e inserendo nel msg error, il messaggio ricevuto, possiamo intercettarlo in fase di debug.

Altro semplice aiuto è quello di utilizzare il case insensitive match, disponibile solo dalle versioni 2015 in su.



Case Insensitive match

Un approfondimento sul tema Stringhe vs Enum è discusso qui.

Ci sono altri casi che riteniamo obbligatori quando si implementa una SSM.

Idle

Il caso 'idle' è un caso di attesa a fare un operazione, lo approfondiremo nel prossimo articolo sull' utilizzo della Event Strucure. In questo caso si

dovrebbe mettere sempre una logica di attesa di un determinato evento o un attesa di input in generale.

Init

Il caso Init inserisce le logiche di apertura di riferimenti, e tutte le operazioni che devono essere eseguite in una fase di inizializzazione.

Il fatto di inserirlo come stato all'interno della SM, ci offre la possibilità di richiamarlo anche durante l'esecuzione.

Error

Centralizziamo gli errori per poter eseguire in unico spazio il codice di gestione e le azioni da eseguire in caso di errore. Questo viene combinato con una case di ErrorTrap da posizionare in uscita alla struttura case principale e che devia il messaggio ad un messaggio di Error.

Exit

In questo stato gestiamo un uscita con CleanUp e tutte le operazioni di chiusura riferimenti ecc.

Quindi i casi da inserire sempre sono 5 **I**nit,**I**dle,**D**efault,**E**rror,**E**xit

Un implementazione di macchina a stati è la JKI State Machine, che potete scaricare da qui, utilizzando il VI Package Manager.

Nella prossima parte vedremo la funzionalità della Event-Based Simple State Machine