

LabVIEW DVR e Functional Global (Action Engine) a confronto

Category: LabVIEW Talk

scritto da admin | Maggio 29, 2025



Introduzione

Le [Functional Global Variables \(FGV\)](#), inizialmente conosciute come LV2Globals, rappresentano un'evoluzione rispetto alle variabili globali classiche in LabVIEW, nate per risolvere il problema dell'accesso concorrente ai dati condivisi.

A differenza delle variabili globali native, le FGV sono costruite attraverso VI non rientranti, i quali, proprio per questa caratteristica, garantiscono l'accesso controllato alla risorsa condivisa.

Quando una FGV è in uso, le altre istanze vengono bloccate fino al termine dell'esecuzione corrente, riducendo così le possibilità di corruzione dei dati, ma inevitabilmente introducendo un limite prestazionale.

Il potenziale delle FGV risiede nel fatto che possono contenere codice: questa possibilità ha spinto molti sviluppatori a utilizzarle non solo come semplici contenitori di dati (con operazioni di get e set), ma come entità intelligenti, capaci di gestire logiche più complesse.

Tra gli utilizzi, l'implementazione di variabili WORM (Write Once Read Many), dove l'accesso in scrittura viene limitato a una sola occasione, lasciando la lettura libera.

Queste soluzioni si sono evolute nel concetto di Action Engine (AE), un paradigma ispirato ai linguaggi orientati agli oggetti.

Una AE è sostanzialmente una FGV arricchita da un enumeratore typedef che funge da selettore di azioni (metodi), e da un cluster typedef che incapsula lo stato (proprietà) della "pseudo-classe".

Tale struttura consente di simulare un comportamento simile a quello di una

classe singleton, dove l'accesso ai dati è centralizzato e gestito in modo controllato.

Gli AE possono essere incapsulati in VI impostati come inline, i quali espongono solo l'azione desiderata con il terminale necessario.

Questo approccio riduce il sovraffollamento dei connettori e permette una maggiore modularità.

Prima dell'introduzione della programmazione orientata agli oggetti in LabVIEW, gli AE rappresentavano la soluzione di riferimento per modellare entità con stato persistente e comportamenti associati.

Nonostante la presenza delle classi native, molte realtà, specialmente nel mondo del Test e Misura, continuano a utilizzare le FGV/AE grazie alla loro semplicità di implementazione, alla minore curva di apprendimento e alla maggiore immediatezza rispetto all'OOP.

Questo è particolarmente vero in contesti in cui la priorità è l'efficienza nello sviluppo e nella manutenzione, più che la purezza architetturale del codice.

Con l'evoluzione delle esigenze applicative, soprattutto in contesti ad alta concorrenza e con grandi volumi di dati, è emersa la necessità di un meccanismo più efficiente per la gestione dello stato condiviso.

È in questo contesto che sono stati introdotti i Data Value References (DVR).

I DVR permettono di creare riferimenti a dati tipizzati, consentendo un accesso per riferimento anziché per valore.

Questo approccio riduce la duplicazione dei dati e migliora le prestazioni, soprattutto quando si lavora con strutture dati di grandi dimensioni o in ambienti multi-threaded, utilizzando la struttura In Place Element (IPES).

I [DVR](#) si integrano perfettamente con la programmazione orientata agli oggetti in LabVIEW, permettendo la creazione di classi che gestiscono internamente lo stato attraverso riferimenti, migliorando la modularità e la scalabilità delle applicazioni.

Inoltre, l'utilizzo delle strutture In Place Element garantisce un accesso sicuro e controllato ai dati, prevenendo condizioni di race e migliorando la robustezza del sistema.

In questo articolo cerchiamo di mettere un ordine ai vari nomi e raccogliamo la nostra esperienza e le discussioni di Community come LAVA, NI.com e articoli di vari autori di blog LabVIEW.

Functional Global Variable (FGV)

Le FGV (qualcuno le ha anche nominate LV2global) sono VIs non rientranti che utilizzano shift register non inizializzati per mantenere lo stato **interno** tra invocazioni successive e una struttura Case per creare funzioni di Get e

Set o altre come Init.

Questo pattern consente di centralizzare i dati condivisi e incapsulare le operazioni di accesso, come lettura e scrittura.

Pro

Persistenza dello stato senza globali.

Serializzazione implicita grazie alla non rientranza.

Semplicità di implementazione.

Rispetto alle variabili Global tradizionali, permettono codice di protezione (es. WORM Write Once Read Many) o monitoraggio e log per debugging.

Contro

Concorrenza limitata: **ogni chiamata blocca la FGV.**

Prestazioni limitate: Anche se limitato la race condition, rispetto alle variabili Globali pure sono meno performanti.

Immaginate che essendo un unico VI non rientrante ogni chiamata indipendentemente, blocca le altre.

Action Engine (AE)

L'AE nasce come evoluzione concettuale della FGV. È un VI con una struttura case all'interno di un ciclo While con shift register non inizializzati, ma con una logica strutturata in "azioni" selezionabili, come Read, Write, Reset, Accumulate, ecc.

Architettura classica

L'AE funge da motore centrale per lo stato e la logica.

Spesso viene incapsulato in una libreria di VIs (LLB o libreria virtuale), dove il core AE è invocato da VIs wrapper dichiarati come **inline**, ciascuno responsabile di una **singola funzione** (es. Get Value.vi, Set Value.vi, Add Item.vi).

Questo approccio ha diversi vantaggi:

Riduzione della complessità del pannello connettore dell'AE.

Esplicitazione delle interfacce secondo il principio di responsabilità singola.

Utilizzo di cluster typedef per raggruppare parametri, riducendo il numero di terminali e facilitando la manutenzione.

Storicamente...

Prima dell'introduzione della programmazione orientata agli oggetti (LV00P), questo era il metodo preferito per simulare una **classe singleton** in LabVIEW: un'istanza unica del core AE gestisce tutto lo stato, con VIs inline che fungono da metodi della "classe".

Tabella comparativa

Caratteristica	FGV	AE	DVR
Accesso ai dati	Per valore	Per valore	Per riferimento
Persistenza	Shift register	Shift register	Reference in heap
Serializzazione	Implicita (non reentrante)	Implicita (non reentrante)	Esplicita (lock con IPE)
Scalabilità	Limitata	Media	Alta
Parallelismo	Limitato	Limitato	Elevato
Estendibilità	Limitata	Alta con VIs inline	Alta (combinabile con LV00P)
Complessità di implementazione	Bassa	Media	Alta
Uso in architetture moderne	Legacy	Spesso come core-logica	Standard in OOP e actor-model

Quando scegliere cosa

Scenario	Pattern consigliato
Accesso semplice a uno stato condiviso (contatore, flag, ecc.)	FGV
Gestione centralizzata dello stato con più operazioni	AE con VIs inline
Accesso concorrente efficiente e strutture dati complesse	DVR (eventualmente in LV00P)

Conclusione

Non esiste un pattern "migliore" in senso assoluto: ciascuno ha uno spazio d'uso ben definito.

Le FGV sono rapide da implementare e ideali per casi semplici.

Le AE, se incapsulate con criterio e modularizzate tramite VIs inline, rappresentano una forma solida di architettura modulare, ideale per logiche sequenziali.

I DVR, infine, aprono la porta alla programmazione orientata agli oggetti, alla concorrenza e a soluzioni scalabili in sistemi complessi.

Risorse consigliate

- [Functional Global Variable – Bytelabs](#)
- [LabVIEW DVR e Functional Global a confronto – Bytelabs](#)
- [Application Engines in LabVIEW – LinkedIn](#)
- [NI White Paper: DVR e Synchronization Structures](#)
- [VI Shots – TS2147: Advanced Design Patterns in LabVIEW](#)