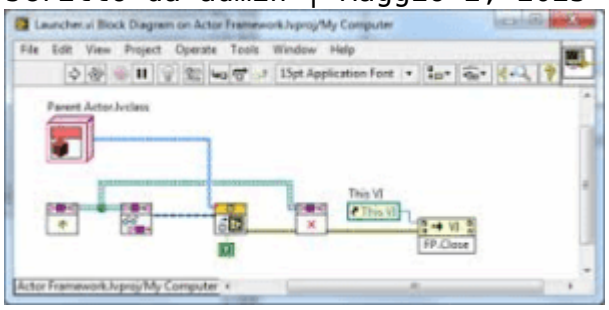
Guida alla Scelta di Pattern e Framework per Sviluppatori LabVIEW

Category: LabVIEW Talk
scritto da admin | Maggio 2, 2025



Perché parlare di pattern e framework?

La programmazione in LabVIEW, specie nei contesti industriali e di sviluppo software avanzato, esce rapidamente dalla fase sperimentale in cui semplici VI non organizzati vanno bene, per entrare in quella di produzione stabile, scalabile e manutenibile (Software professionale, Test Bench automatizzati e complessi).

L'adozione di pattern e framework non è solo una questione "tecnica" o di moda: rappresenta la risposta strutturata a sfide fondamentali come:

Performance e affidabilità: framework collaudati permettono di gestire in modo efficiente l'esecuzione concorrente, il consumo di risorse e il recupero da errori.

Gestione della complessità: i progetti di test e misura spesso includono interfacce hardware multiple, logiche di controllo articolate e requisiti di affidabilità.

Senza una solida architettura, il codice diventa fragile e difficile da comprendere o modificare.

Manutenibilità: software scritto senza una chiara struttura diventa nel tempo un "mostro" da manutenere, specie quando a lavorarci sono team diversi o quando si devono integrare nuove funzionalità.

Riutilizzabilità: pattern e framework incoraggiano la scrittura di moduli generici e ben definiti, che possono essere usati in progetti diversi riducendo tempi e costi di sviluppo.

Standardizzazione: garantiscono una base comune di sviluppo all'interno di un team, facilitando la collaborazione e l'onboarding di nuovi sviluppatori.

Qualità e testabilità: strutture coerenti e ben progettate si prestano naturalmente all'implementazione di test automatici, alla verifica della correttezza e al debugging.

Framework LabVIEW: cosa sono e perché usarli

I framework sono “scatole” di codice già predisposte che implementano pattern consolidati, includendo strumenti per la comunicazione, la gestione degli errori, la configurazione, la documentazione, e molto altro.

Vantaggi:

- Accelerano lo sviluppo, fornendo una base solida;
- Garantisco una struttura uniforme;
- Offrono moduli riutilizzabili;
- Spesso includono tool di debugging e logging integrati.

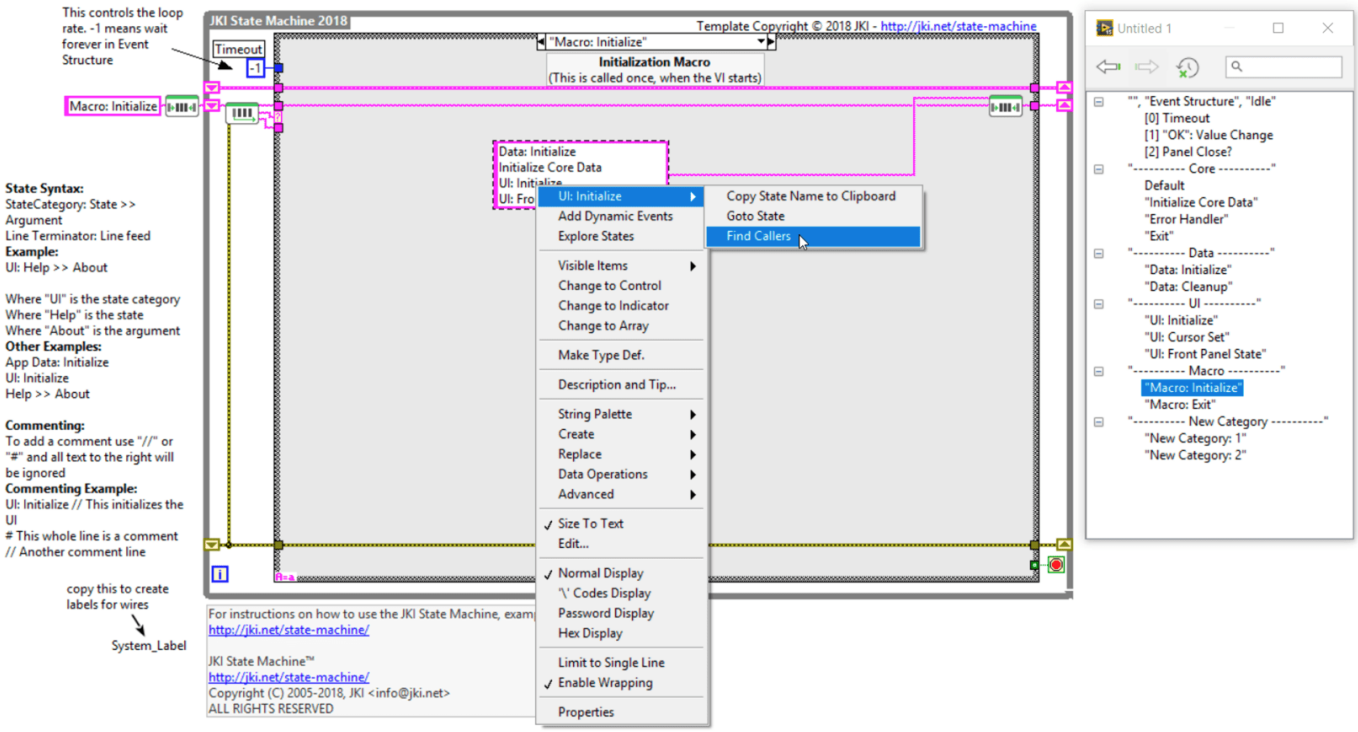
Obiettivi principali nell’utilizzo di pattern e framework in LabVIEW

- **Organizzare il codice in modo modulare e gerarchico**
  Separare le responsabilità del software in moduli ben definiti, ciascuno con compiti chiari e interfacce semplici.
- **Separare la logica di controllo dalla logica di presentazione (UI)**
  Permette di modificare interfacce utente senza impattare la logica di fondo e viceversa.
- **Gestire in modo efficace gli eventi e la concorrenza**
  Utilizzare pattern asincroni come Producer/Consumer o Actor Framework per migliorare reattività e affidabilità.
- **Favorire il riuso e la scalabilità**
  Scrivere codice generico, estendibile e facilmente adattabile a contesti diversi, evitando duplicazioni.
- **Facilitare il debug e la manutenzione**
  Implementare meccanismi di logging, gestione degli errori e test unitari integrati.
- **Promuovere la documentazione e standardizzazione interna**
  Favorire l’adozione di convenzioni di codifica, naming e organizzazione del progetto.

Integrazione con la panoramica dei framework

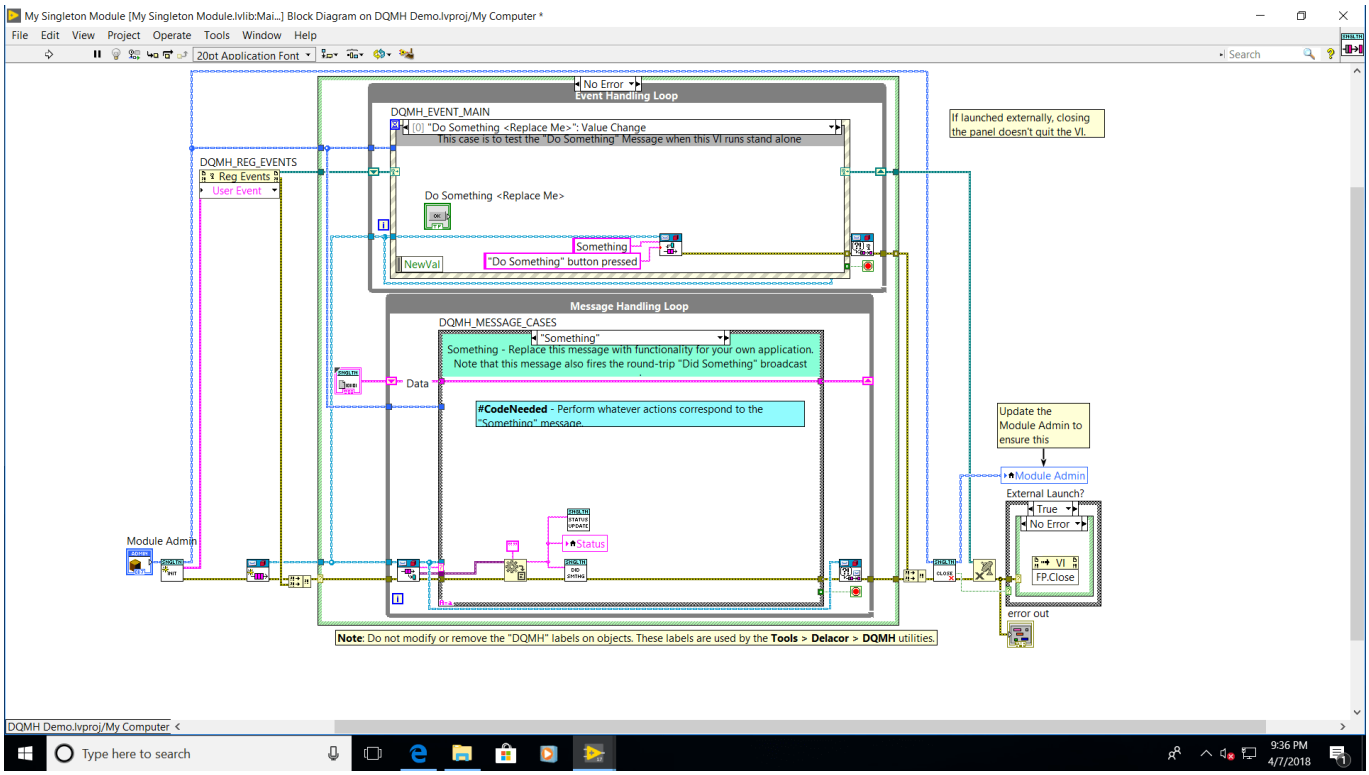
Come anticipato nella panoramica precedente, la scelta del framework si basa sugli stessi principi e obiettivi. Ad esempio:

JKI State Machine



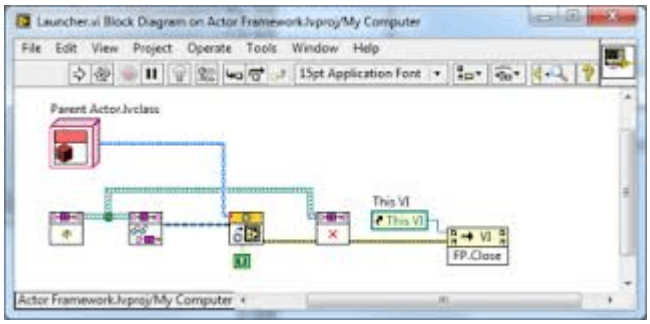
Semplifica la gestione della UI e degli stati, facilitando la separazione logica e la manutenibilità.

Delacor DQMH



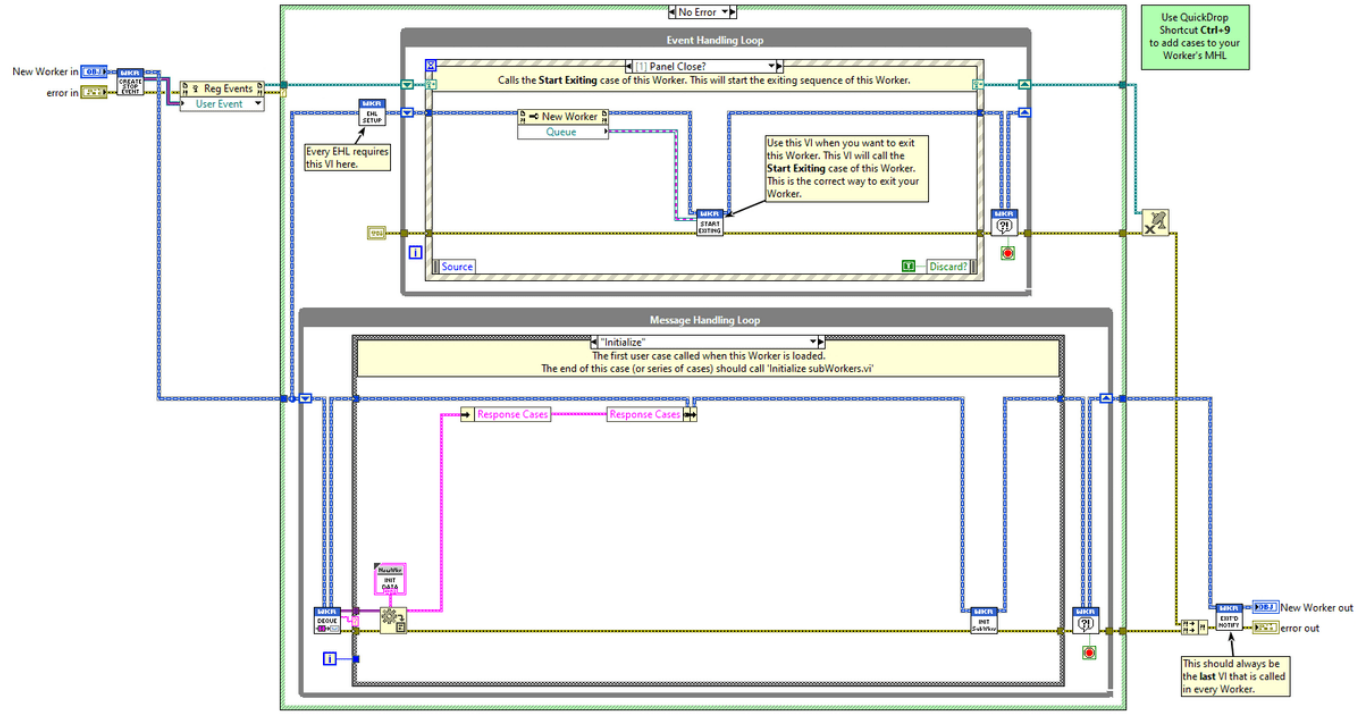
Aggiunge modularità e supporto per test automatizzati, ideali per team e progetti complessi

Actor Framework



Permette la massima scalabilità e gestione concorrente, ma richiede un investimento formativo maggiore.

Worker for LabVIEW



Offre un buon compromesso tra modularità e complessità.

QMH

è un ottimo punto di partenza per sistemi semplici e medi.

Customized Framework

In Bytelabs, abbiamo sviluppato framework personalizzati che adottano queste best practice, unendo liobrerie e metodi che abbiamo creato negli anni , focalizzandoci sull’estendibilità, la standardizzazione e il supporto agli strumenti di test.

Best Practice per l’adozione di pattern e framework

Analisi preliminare

- Valutare la complessità e dimensione del progetto;
- Identificare i requisiti funzionali e non funzionali;
- Stimare l’orizzonte temporale e le risorse disponibili.

Selezione consapevole del pattern/framework

- Non adottare framework complessi se non necessari;
- Considerare la curva di apprendimento e la formazione del team;
- Preferire framework con supporto e documentazione solida.

Architettura modulare e OOP

- Usare classi LabVIEW per incapsulare dati e funzioni;
- Separare UI, logica di controllo e gestione dati.

Gestione asincrona e messaggistica

- Usare code e messaggi per disaccoppiare moduli;
- Evitare dipendenze rigide.

Centralizzare gestione errori e logging

- Implementare un sistema uniforme di rilevazione, propagazione e risoluzione degli errori;
- Loggare eventi importanti per debug e audit.

Documentazione e naming

- Mantenere standard di nomenclatura chiari;
- Commentare il codice per facilitare la comprensione.

Testing e Debug

- Integrare test unitari e di sistema;

- Usare strumenti di profiling per ottimizzare performance.

Formazione continua

- Aggiornare le competenze del team;
- Condividere esperienze e best practice.

Casi concreti e raccomandazioni per Bytelabs

Nel contesto industriale e di test & misura, Bytelabs consiglia:

- Progetti semplici: SSM Simple State Machine o ESM
- Per progetti medio-piccoli: JKI State Machine o QSM;
- Per progetti di medio-grande scala, distribuiti o con elevata concorrenza: QMH, Actor Framework, Delacor DQMH;
- Investire in formazione dedicata per Team;
- Implementare una libreria interna di componenti riutilizzabili e pattern standard;
- Integrare sempre i processi di test unit, logging e gestione errori.

4. Principi dell'ingegneria del software

Pattern, architetture e framework incarnano principi quali **SOLID**, **separazione delle responsabilità** e **riusabilità**, consentendo:

1. **Prefabbricazione** di componenti per velocizzare lo sviluppo .
2. **Modularità** ed **estendibilità** con OOP e generation tool.
3. **Focus** sul problem solving, non sui dettagli implementativi .

6. SPOT: il nostro framework custom

In Bytelabs utilizziamo spesso QMH come architettura di base e librerie sviluppate internamente che lo rendono una architettura customizzata e in cui ci troviamo comodi per una buona parte di progetti.

Avendo due architect nel nostro staff, abbiamo la tendenza do sviluppare nostri framework e architetture, sicuramente partendo dallo studio di altri e cercando di prendere spunto.

Facciamo questo perché la complessità non è semplice da gestire e usare qualcosa che non conosciamo profondamente, ci porta ad errori e debug acrobatici solo per la complessità del codice del framework stesso.

Tuttavia abbiamo lavorato spesso con la macchina a stati di JKI, ci piace e lo abbiamo usato il DQMH o anche su Actor Framework che però inizia ad avere una profondità che porta complessità e articolazioni troppo elevate e qualche volta in contraddizione con il nostro modello di sviluppo.

Anche Worker è sicuramente molto interessante ma ancora non è capitata l'occasione di utilizzarlo.

In Bytelabs nel tempo abbiamo sviluppato progetti template, librerie e framework di partenza nostri, Come ByteQMH una QMH interamente custom con librerie incluse, ByteAF un nostro Actor Model per progetti che richiedono scalabilità e in generale una progettazione Object Oriented.

Come consulenti proponiamo Framework e aiutiamo ad implementare all'interno dei progetti, sia con la formazione Learning By Doing, sia con corsi a tema, con scaletta organizzata sui vostri obiettivi.

SPOT (System Plugin Optimize Test) è il **framework** di Bytelabs, concepito per:

- **Interfacciamento database**: supporto nativo per SQLite e SQL/ODBC.
- **Editing no-code**: configurazione visuale di flussi di test e misure.
- **Classi di interfaccia**: estendibilità rapida scrivendo soltanto azioni specifiche.
- **Prefabbricazione** delle funzionalità di base, riducendo tempi e costi di sviluppo, aumentando efficacia, competitività e soddisfazione dei clienti.

SPOT rappresenta la nostra visione di **framework custom**: architetture pensate su misura per team e applicazioni, integrate in un ecosistema multilinguaggio e modulare.

Considerazioni finali

L'adozione consapevole di pattern e framework non è una spesa, ma un investimento strategico che porta benefici evidenti nel medio-lungo termine, in termini di:

- riduzione dei tempi di sviluppo;
- miglior qualità e affidabilità del software;
- facilità di manutenzione e aggiornamenti;
- aumento della soddisfazione del cliente e del team.

Approfondimenti

Nome Framework	Descrizione Breve	Sito Ufficiale / Documentazione	Download VIPM	Altri Link Utili
Delacor DQMH	Framework modulare basato su coda e eventi	https://dqmh.org/	https://www.vipm.io/package/delacor_lib_dqmh_toolkit/	https://www.youtube.com/playlist?list=PLmF-6jvwRvVOpNdPcRzskvnGbW79ug6AZ
Actor Framework (AF)	Orientato agli attori per sistemi complessi	https://labviewwiki.org/wiki/Actor_Framework	Incluso in LabVIEW o su GitHub	https://github.com/ni/actor-framework
JKI State Machine	State machine avanzata e riutilizzabile	https://www.jki.net/blog/topic/state-machine	https://www.vipm.io/package/jki_lib_state_machine/	https://www.jki.net/labview-training
Workers for LabVIEW	Framework modulare, dinamico, basato su eventi	https://workersforlabview.gitbook.io/	https://www.vipm.io/package/sc_workers/	https://www.scarfecontrols.com/
SPOT (Bytelabs)	Sequencer modulare per LabVIEW	https://www.byteqx.com/en/spot-test-sequencer/	Personalizzato, integrato con Bytelabs	https://www.linkedin.com/posts/bytelabs_spot-preview-activity-7249407088936837122-Ys75