Utilizzo delle nuove tecnologie per l’analisi di codice.

Category: Insight
scritto da Lorenzo Borghi | Novembre 29, 2024

Negli ultimi anni, l’**intelligenza artificiale** si è affermata come un supporto estremamente utile per facilitare lo svolgimento di **attività complesse**, dalla ricerca di informazioni, fino ad arrivare alla pianificazione e organizzazione di attività. AI più avanzate sono in grado di **generare bozze di script** che agevolano i programmatori e riducono significativamente i tempi di sviluppo.

Tuttavia, questi strumenti non sono ancora del tutto autonomi nella produzione di contenuti di qualità: un occhio attento, può spesso notare pattern tipici delle reti neurali, sintomo di un approccio ancora procedurale. Nonostante ciò, è possibile utilizzare questi potenti alleati per il compito contrario, non quello di **scrivere del codice**, ma di **aiutare a comprenderlo**, spiegandone le funzionalità e organizzandolo in blocchi facili da leggere.

Un esempio pratico è l’utilizzo delle AI per **interpretare linguaggi di programmazione** un po’ **datati** come **Visual Basic**, composti spesso da enormi blocchi di testo monoliti, intricati e difficili da analizzare.
In questi casi, un’intelligenza artificiale, prendendo una delle più famose, **ChatGPT**, si rivela essere un compagno estremamente utile, specialmente se il codice dev’essere migrato verso un ambiente completamente diverso.

Un interessante caso di studio da analizzare, riguarda proprio la conversione di un software scritto in **VB6** in **LabVIEW**. Qui emerge un’importante distinzione: mentre le **AI** possono essere di valido aiuto nella comprensione di codice di testo, il loro utilizzo per comporre correttamente dei VI (Virtual Instruments) in LabVIEW si rivela spesso inaffidabile, a causa di informazioni spesso errate, o su blocchi inesistenti.

Contesto Operativo

La migrazione di per sé non rappresenta un’impresa insormontabile, ma la fase di analisi del codice costituisce una parte estremamente prolissa del progetto. Affrontare questa fase in modo esaustivo all’interno di un singolo articolo sarebbe troppo dispersivo.

Un esempio pratico può aiutare a chiarire.

Considerando una variabile denominata **lUWSampleVar**, dichiarata nel codice come *Long*, il prefisso **lUW** indica che si tratta di un *Long Unsigned Word* con un peso di 16 byte. Tuttavia, questa variabile viene collocata in uno slot di tipo *Long* standard, che secondo la documentazione di VB6 ha un peso di 32 byte.

La situazione si complica ulteriormente quando il codice contiene diversi prefissi, ciascuno con una dimensione specifica. Alcuni, come **dUDW** (*Double Unsigned Double Word*, 64 byte), rappresentano variabili più grandi, mentre altri, come **bUB** (*Byte Unsigned Byte*, 1 byte), risultano decisamente più piccoli.

L'unico riferimento utile era costituito da una mappatura delle variabili, che associava i prefissi al peso corrispondente. Tuttavia, in alcuni casi, questa mappatura poteva risultare ambigua: ad esempio, il prefisso **lDW** (*Long Double Word*), che indicava un dato da 4 byte, poteva essere confuso con la sua versione da 8 byte, generando ulteriori complicazioni.

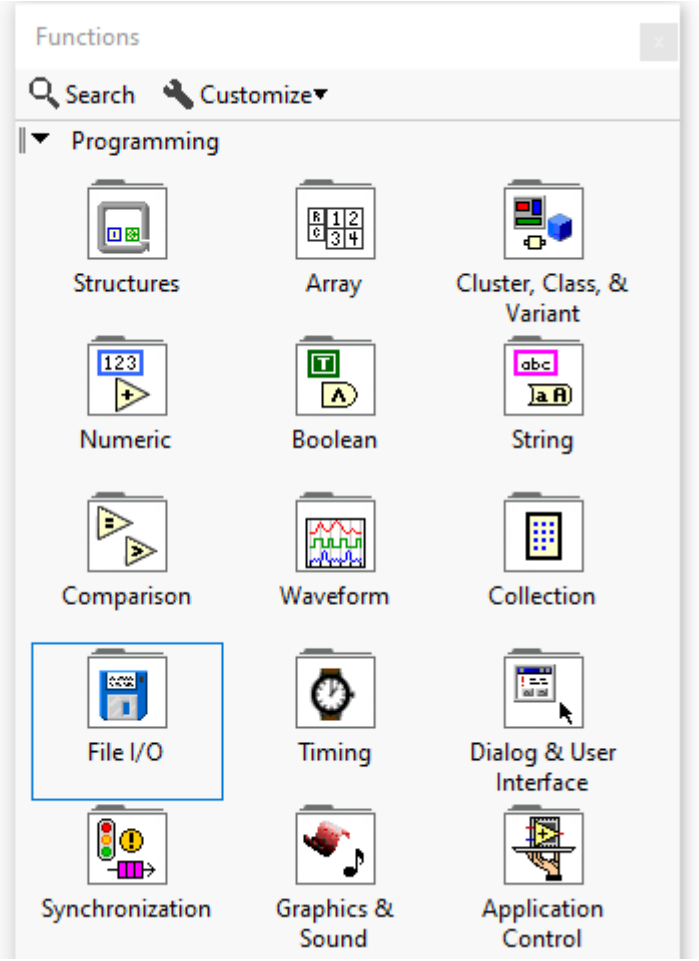
A rendere l'analisi ancora più complessa era l'utilizzo di Visual Studio, che per VB6 non applicava alcuna colorazione al codice per distinguere i diversi tipi di testo. Questo comportava la visualizzazione di un enorme muro di testo monocromatico, rendendo il processo particolarmente laborioso.

Per comprendere l'utilità di questa analisi, è necessario spostare l'attenzione su LabVIEW e approfondire il processo di estrazione dei dati da un file, nel caso osservato, un file binario.

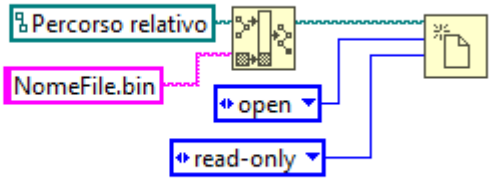
Lettura di dati binari in LabVIEW

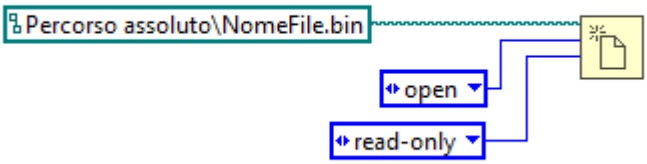
Apertura del File

Il primo passo consiste nell'**apertura del file binario**. Per farlo, è necessario creare un nuovo V**irtual Instrument (VI)** e accedere alla palette di gestione dei file, selezionando l'opzione *Open/Replace/Create File*.



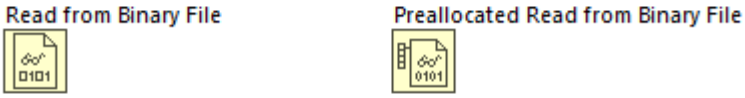
Questa funzione permette di aprire un file in **modalità lettura** o scrittura. In questo caso, abbiamo bisogno solo della modalità lettura, specificando un percorso assoluto o relativo per il file da aprire.





Funzioni di lettura

Una volta aperto il file, si passa alla lettura dei dati; LabVIEW offre due modalità principali per questo scopo: **Read from Binary File** e **Preallocated Read from Binary File**.



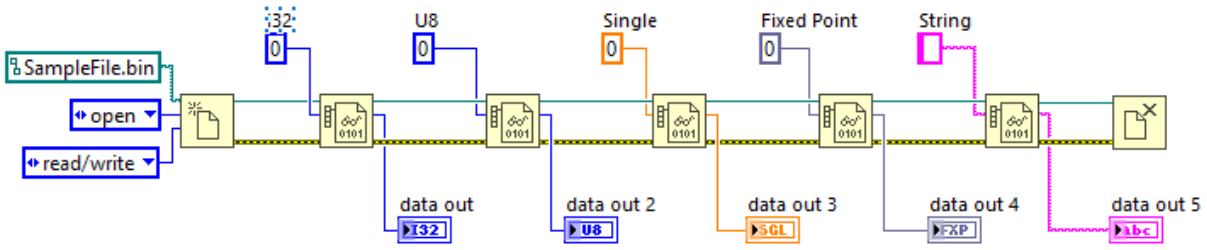
La principale differenza tra queste due modalità risiede nella gestione della memoria: la prima modalità alloca dinamicamente memoria e copia i dati in un nuovo array, mentre la seconda evita tale copia, leggendo direttamente in un array pre-allocato. La scelta tra le due dipende principalmente dalle dimensioni del file da leggere. A questo punto, configurando correttamente le variabili numeriche, è possibile determinare il numero di byte da estrarre dal file.

Inoltre, il terminale di ingresso varia tra le due funzioni: nella prima modalità, è possibile collegare sia un percorso di file, ignorando l'operazione di apertura del file di cui si parlava in precedenza, sia un riferimento al file stesso; nella seconda, invece, è possibile collegare solo il riferimento al file.

Una volta concatenati i blocchi di apertura e lettura, i dati estratti possono essere impiegati nei processi successivi del sistema, consentendo una **gestione accurata** e precisa delle variabili ottenute dal file binario.

Estrazione dei dati

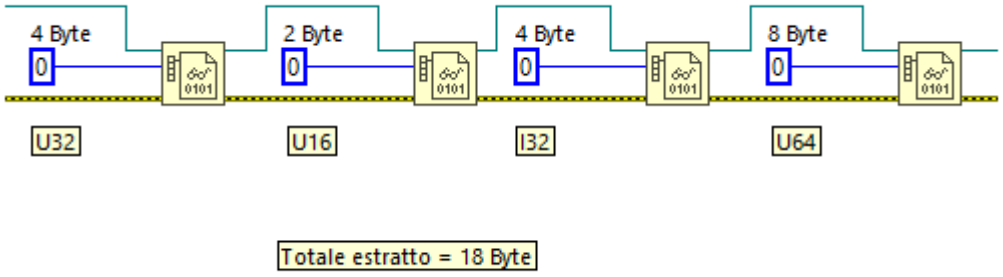
L'immagine seguente è fornita a scopo illustrativo per mostrare come la funzione si comporta se si collegano terminali con **dati differenti in ingresso**. È importante notare che l'accesso parallelo concorrente al file, tramite la stessa reference, potrebbe generare errori di lettura. Pertanto, si raccomanda di eseguire una **lettura sequenziale** dei dati, come mostrato.



Al terminale di ingresso del tipo di dato (**Data In**) è possibile inserire variabili numeriche di dimensioni diverse, consentendo di estrarre un numero variabile di byte dal file binario. Il terminale di uscita, denominato **Data Out**, restituirà valori dello stesso tipo del dato in ingresso, assicurando coerenza tra input e output durante il processo di lettura.

Una volta configurati in sequenza i blocchi per l'apertura e la lettura del file, è possibile leggere i valori estratti e utilizzarli nei passaggi successivi del processo. È importante sottolineare che la lettura del file avviene in modo sequenziale; pertanto, il cavo che trasmette il riferimento al file deve essere collegato da un blocco di lettura all'altro. Il parallelismo mostrato nell'immagine precedente è puramente dimostrativo e serve a illustrare graficamente la coerenza nel tipo di dato selezionato tra ingresso e uscita.

Una scelta accurata del tipo di dato è **fondamentale** per garantire un'**adeguata estrazione** dal file. Poiché la lettura avviene in modo sequenziale, il peso in byte di un dato estratto in un punto iniziale influenzerà tutte le letture successive, con il rischio di causare un effetto a catena. Questo potrebbe generare letture errate e restituire valori non corretti per tutti i dati estratti successivamente, compromettendo l'intero processo.



Analisi del VB6 tramite IA

Le problematiche del codice.

Comprendere come è stata affrontata questa migrazione aiuta a coglierne la complessità e le soluzioni adottate per superare le difficoltà. L'incarico era quello di estrarre e mappare oltre 400 variabili da un blocco di dati binari di 6 Kb. Ogni variabile doveva essere trattata seguendo una mappatura dettagliata fornita in anticipo.

Uno dei problemi principali era legato alla natura del codice originale, scritto in VB6; senza disporre della suite proprietaria per la compilazione, non era possibile mandare in Run il codice, per osservare i valori dei dati estratti in tempo reale.

L'eseguibile esistente mostrava solo una piccola parte delle variabili tramite l'interfaccia, rendendo impossibile il confronto diretto tra i valori estratti in LabVIEW e quelli gestiti dal codice originale.

L'utilizzo di ChatGPT

A questo punto, l'utilizzo di ChatGPT, che si è rivelato essenziale per affrontare le difficoltà incontrate durante il processo; tramiti specifici comandi si è stati in grado di trasformare l'intelligenza artificiale in un assistente capace non solo di monitorare l'avanzamento del lavoro, ma anche

di semplificare la gestione della complessa mappatura delle variabili.

Man mano che venivano analizzate le righe di codice VB, ChatGPT, che riceveva come input la sequenza delle variabili estratte, memorizzava i dati, li organizzava in modo coerente e li associava ai corrispondenti pesi in byte, che venivano calcolati in base ai tipi di dato presenti nella mappatura originale.

Questo approccio, che eliminava la necessità di effettuare continuamente conversioni manuali dalla notazione dei tipi di dato al calcolo dei pesi, consentiva di ottenere una lista già pronta, ordinata e facilmente utilizzabile per l'implementazione nel VI di LabVIEW, evitando così errori ripetuti e accelerando notevolmente il processo.

La versione 4o di ChatGPT, grazie alla capacità di memoria intrachat, ha permesso di lavorare in modo più fluido: si potevano fornire dati all'intelligenza artificiale che venivano poi utilizzati nei passaggi successivi. Questo ha facilitato l'applicazione di schemi logici semplici su grandi quantità di dati.

Il processo di analisi

Nel concreto, il processo avveniva così: si forniva un blocco di codice VB, ad esempio con 5 variabili….

```
Get #lngFileIn, , .lUBSample1
Get #lngFileIn, , .dUWSample2
Get #lngFileIn, , .lUWSample3
Get #lngFileIn, , .bUBMaxSample
Get #lngFileIn, , .bUBMinSample
```

…venivano controllati i tipi di dato associati alle variabili…

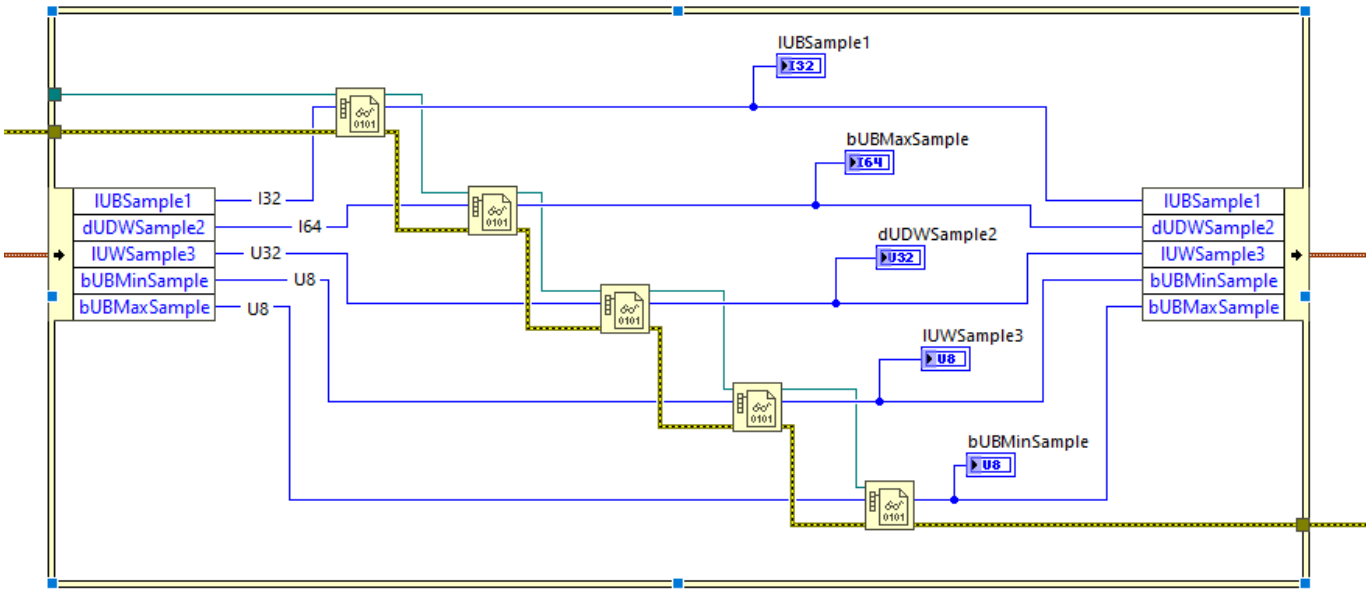
```
Dim lUBSample1 As Long
Dim dUWDSample2 As Double
Dim lUWSample3 As Long
Dim bUBMaxSample As Byte
Dim bUBMinSample As Byte
```

…e, tramite specifici prompt, ChatGPT teneva traccia del progresso, aggiornando in tempo reale una lista con il peso totale dei dati estratti.

- **lUBSample1** è di tipo lUB, che occupa *4 byte*.
- **dUDWSample2** è di tipo dUDW, che occupa *8 byte*.
- **lUWSample3** è di tipo lUW, che occupa *4 byte*.
- **bUBMinSample** è di tipo bUB, che occupa *1 byte*.
- **bUBMaxSample** è di tipo bUB, che occupa *1 byte*.

Somma: 4 + 8 + 4 + 1 + 1 = *18 byte* complessivi.

Una volta verificata la correttezza delle informazioni, queste venivano riportate nel VI di LabVIEW, utilizzando il meccanismo di estrazione *Preallocated Read from Binary File* illustrato precedentemente.



Anche se in rare occasioni era necessario correggere ChatGPT, l'approccio ha notevolmente velocizzato il lavoro.

L'interazione continua con l'IA ha permesso di superare i limiti del metodo *trial and error*, portando a una migrazione rapida ed efficiente.

Il risultato finale? Un processo complesso che, grazie alla collaborazione con ChatGPT, è stato reso più fluido, riducendo sensibilmente i tempi e gli sforzi necessari.