GESTIRE I CAMBIAMENTI MEDIANTE LA PROGRAMMAZIONE ORIENTATA AGLI OGGETTI IN LABVIEW

Jan Klasson

L'articolo utilizza un caso esemplificativo per esplorare l'effetto della conversione di un programma LabVIEW tradizionale in un programma orientato agli oggetti

A seconda di come il vostro programma LabVIEW è strutturato, lo sforzo necessario per modificare ed estendere il programma stesso può variare notevolmente. La programmazione orientata agli oggetti, introdotta in LabVIEW 8.20, è una tecnica che può aiutarvi a migliorare la struttura del programma e a semplificare l'aggiunta e la modifica di funzionalità.
Lo scopo è quello di sottolineare come l'orientamento agli oggetti influisca sulla struttura del programma ed esplorare i benefici riguardanti le modifiche e le estensioni del programma.

CHE COS'È LA PROGRAMMAZIONE ORIENTATA AGLI OGGETTI?

La programmazione orientata agli oggetti è una tecnica che raggruppa funzionalità e dati insieme in un modulo di codice comune, denominato classe. In LabVIEW, un buon modo per comprendere il concetto di classe è quello di pensare ad un cluster a cui sono collegati dei VI. Quando tale 'cluster' è cablato attraverso il programma, i fili conterranno istanze dei dati del 'cluster' (oggetti), che vengono gestite tramite i VI collegati.

CASO ESEMPLIFICATIVO: UN SISTEMA DI TEST PER CARICABATTERIE

Né la soluzione tradizionale, né la soluzione orientata agli oggetti dovrebbero essere considerate come una soluzione unica od ottimale (per esempio, noterete che non abbiamo utilizzato MAX). Esse sono semplicemente delle soluzioni possibili e ragionevoli, create a scopo di discussione in questo articolo.

IL PROBLEMA

- Eseguire una serie di test per verificare un caricabatterie.
- La stazione di test prova quattro caricabatterie in parallelo.
- Sono utilizzate quattro unità DAQ di due modelli differenti, gli NI USB DAQ 6210 e 6211.
- Test: viene coperto un esempio di test: la misura dell'assorbimento di corrente del caricabatterie.

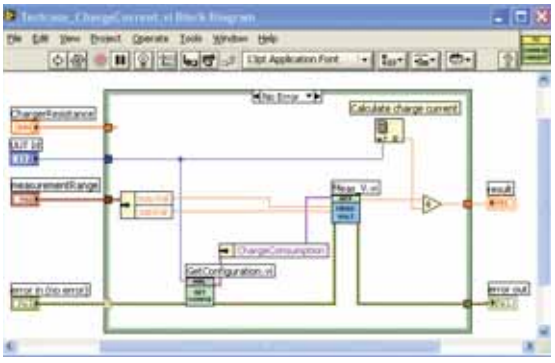

Fig. 1 – Test della corrente di carica strutturato come un programma LabVIEW tradizionale

SOLUZIONE TRADIZIONALE

La figura 1 illustra l'esempio di test, relativo alla misura della corrente di carica.
Il GetConfiguration.vi funziona come una variabile globale, memorizzando la configurazione di tutti i canali fisici di I/O. Meas_V.vi misura la tensione sul canale d'ingresso analogico 'ChargerCurrent'. Poiché lo stesso VI di test viene eseguito per ogni UUT, l'id dell'UUT corrente viene fornito come ingresso al test. La corrente viene calcolata usando la resistenza dell'UUT, a sua volta fornita come array di input al VI dell'esempio di test.

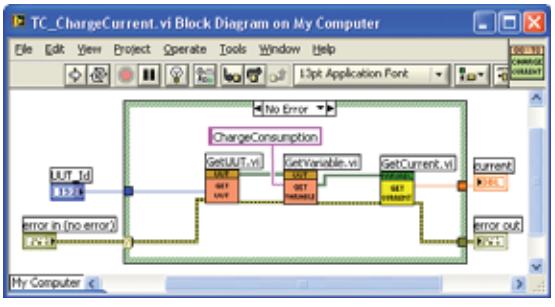

Fig. 2 - Test della corrente di carica strutturato come un programma orientato agli oggetti

SOLUZIONE ORIENTATA AGLI OGGETTI

La figura 2 illustra lo stesso esempio di test modificato in modo da utilizzare un'implementazione orientata agli oggetti. A prima vista, il codice sembra molto simile a quello di un programma LabVIEW tradizionale, ma concettualmente il cambiamento è più grande. I tre subVI appartengono a due classi differenti, UUT e Variable. La finestra di help contestuale, figura 3, indica che GetCurrent.vi appartiene alla classe Variable.lvclass (le classi in LabVIEW sono chiamate lvclass).

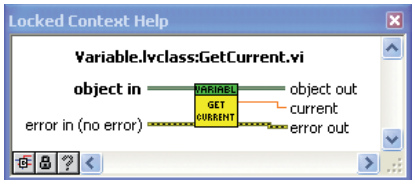

Fig. 3 – Il VI GetCurrent della classe Variable

Il controllo 'oggetto in' contiene l'istanza dei dati (oggetto) della classe che fa lavorare il VI GetCurrent sui dati dell'oggetto collegato.
GetUUT.vi è una variabile globale stile LV 2, che memorizza un oggetto UUT per oggetto di test. In questo esempio, ci sono quattro di tali oggetti. UUT_Id è utilizzato come indice per recuperare l'oggetto corretto. Ogni oggetto UUT contiene tutte le informazioni di configurazione per un UUT.
GetVariable.vi restituisce un oggetto Variable, denominato ChargeConsumption, che rappresenta un certo punto di misura dell'UUT. In questo caso è il punto nel quale viene misurata la tensione del caricabatterie al fine di calcolare la corrente. GetCurrent.vi restituisce la corrente per ChargeConsumption.

Confrontando questo schema a blocchi con l'implementazione tradizionale, potete vedere che la soluzione orientata agli oggetti è meno affollata di dati e più focalizzata su ciò che il test deve fare. L'array di resistenze degli UUT non è più necessario, perché questa informazione viene letta da un ini-file dall'oggetto UUT ed è quindi utilizzata per inizializzare l'oggetto Variable ChargeCurrent con i dati corretti. Inoltre, l'algoritmo per il calcolo della corrente viene gestito da un VI, anziché esplicitamente nello schema a blocchi, migliorando l'affidabilità.

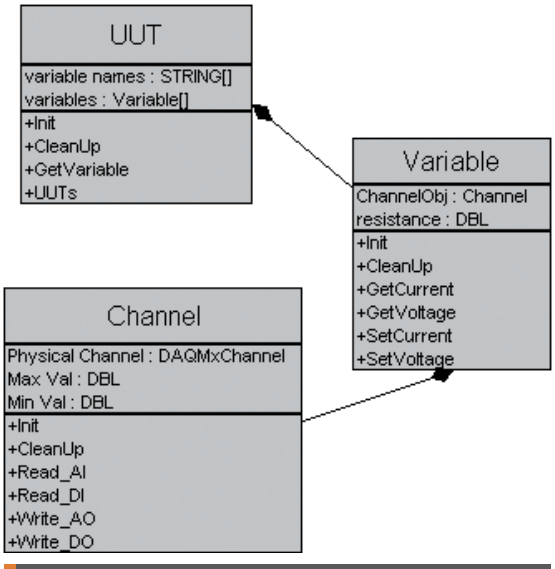

Fig. 4 – Visione d'insieme delle classi della soluzione orientata agli oggetti

Anche i valori min/max ed il riferimento del canale fisico sono scomparsi dallo schema a blocchi nella soluzione orientata agli oggetti. Per spiegare che cosa è successo con tale informazione, nella figura 4 è illustrata una visione d'insieme della struttura del programma orientato agli oggetti.

La figura 4 indica le classi della soluzione orientata agli oggetti. Il campo intermedio del blocco di ogni classe indica i campi dati della classe, mentre il campo inferiore indica i VI della classe.
Ogni oggetto UUT memorizza un array di nomi di variabili ed un array di oggetti Variable. In questo esempio, abbiamo visto solo la variabile ChargeCurrent, ma vi sono naturalmente altre variabili nel sistema completo. Ogni oggetto Variable memorizza un oggetto Channel ed anche la resistenza, se essa è richiesta per il punto di misura. Per la resistenza di ChargeCurrent è utilizzata la resistenza degli UUT. Ogni oggetto Channel rappresenta un canale fisico dal quale viene letta (o scritta) la misura. L'oggetto Channel ha anche i valori min/max per essere in grado di configurare il canale fisico.
Per configurare tutti gli oggetti con i dati corretti, l'oggetto UUT legge le informazioni di configurazione da un ini-file ed usa tali informazioni per inizializzare gli oggetti Variable e gli oggetti Channel. Ecco uno schema di come potrebbero apparire le informazioni di configurazione del canale per la variabile ChargeCurrent (per l'UUT1):

```
[VARIABLE_UUT1_ChargeConsumption]
PhysicalChannel=DAQ1/ai0
MaxVal=5
MinVal=1
```

PRIMA MODIFICA: AGGIUNTA DI UN TEST ADDIZIONALE USANDO CHARGECURRENT

Provare scenari diversi per cambiamenti o aggiunte ad un programma è un ottimo modo per valutare un progetto. La prima aggiunta è un nuovo test che avrà bisogno della misura ChargeCurrent.

Che cosa succede con la soluzione tradizionale? Beh, poiché il calcolo della corrente non è stato inserito in un VI vi sono due possibilità. La prima è quella di duplicare il calcolo della

corrente nel nuovo VI di test. Ma questa non è una buona idea, perché è necessario correggere potenziali errori software in due punti. La seconda possibilità è quella di aggiungere un subVI che calcola la corrente prendendo come parametri la tensione e la resistenza. Questa soluzione è migliore.

Che cosa succede nella soluzione orientata agli oggetti? Tutti i VI di utilità richiesti sono già stati creati e possono essere semplicemente utilizzati nel nuovo VI di test. Pertanto, è facile usare la variabile ChargeCurrent nel nuovo VI di test. Non occorre alcun cambiamento di codice.
Notate come la funzionalità (GetCurrent.vi) sia raggruppata insieme ai dati di cui ha bisogno (la resistenza).
Questo è il modo con il quale i programmi orientati agli oggetti sono strutturati; tipicamente fra i VI di una soluzione orientata agli oggetti sono cablati meno dati.
La ragione è che i dati sono memorizzati all'interno degli oggetti. Questo effetto è molto chiaro se si confrontano le figure 1 e 2.

SECONDA MODIFICA: CAMBIAMENTO DEI MODELLI DI DAQ UTILIZZATI

Uno dei DAQ utilizzati viene sostituito con un nuovo modello. Qual è l'effetto sul sistema? Nella prima versione del sistema c'era un'unità DAQ per ogni UUT. Il nuovo modello di DAQ ha più canali e sono necessarie solo due unità DAQ.

Che cosa succede nella soluzione tradizionale? Nella soluzione tradizionale vi sono due problemi. Il primo problema è illustrato nella figura 5: i numeri del modello di DAQ sono utilizzati come indicatori del VI GetConfiguration.

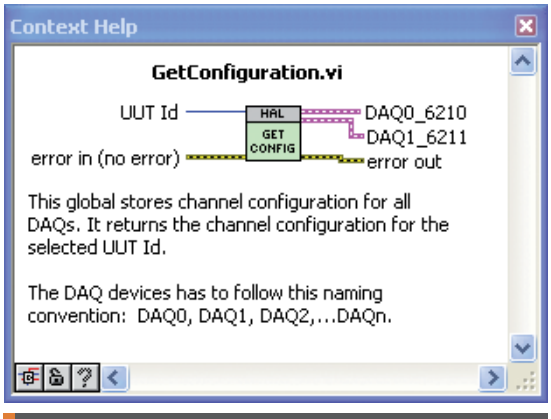

Fig. 5 - Pannello di connessione del VI dati globale GetConfiguration

I nomi degli indicatori devono quindi essere aggiornati; non è un grosso cambiamento, ma è necessario salvare nuovamente il VI di test. Se è stato usato un sistema di controllo della versione, ci sarà una nuova versione del VI. Il secondo problema è evidente quando si legge il testo di help nella figura 5. Per i canali è stata usata una convenzione sui nomi e la funzionalità di questo VI si basa su tale convenzione. Tutti i canali memorizzati in questa variabile globale seguono una convenzione sui nomi come 'DAQ1/ai3'.
Per ottenere il nome di canale di uno specifico id UUT, questo VI sostituisce semplicemente DAQ1 con l'appropriato numero di UUT, per esempio per l'id UUT = 2, il nome del canale sarebbe 'DAQ2/ai3'. L'implementazione di questo VI deve essere riscritta perché possa funzionare nella nuova configurazione hardware. Che cosa succede nella soluzione orientata agli oggetti? Poiché non viene utilizzata alcuna convenzione per i nomi dei canali, è sufficiente aggiornare l'ini-file perché il programma possa funzionare nella nuova configurazione hardware. Non è richiesto alcun cambiamento di codice. Stazioni di test differenti possono utilizzare configurazioni hardware differenti e controllare la configurazione modificando l'ini-file del sistema.

TERZA MODIFICA: AGGIUNTA DI UN MUX PER IL RIUTILIZZO DEI CANALI

Con l'aggiunta di nuovi test, sono necessari altri canali di misura. Pertanto, viene aggiunto un mux al sistema per potere utilizzare i canali in modo più efficiente.

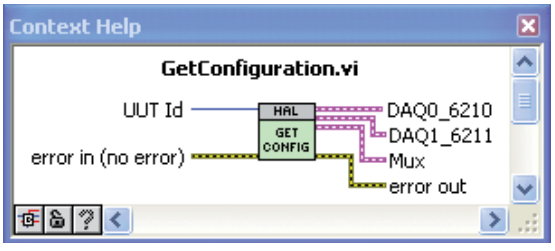

Fig. 6 – Il VI GetConfiguration aggiornato per la gestione del mux.

Che cosa succede nella soluzione tradizionale? Poiché le informazioni di configurazione sono memorizzate nel GetConfiguration.vi, questo è il luogo naturale per aggiungere i dati di configurazione del mux.
La figura 6 illustra il GetConfiguration.vi aggiornato.
Le impostazioni del mux devono essere applicate prima di richiamare Meas_V.vi (che legge l'ingresso analogico).
Che cosa succede nella soluzione orientata agli oggetti? Tre cose, e nessuna di esse ha effetti sul VI di test! In primo luogo, la configurazione del mux deve essere aggiunta all'ini-file. Ecco uno schema di come potrebbe apparire:

```
[VARIABLE_UUT1_ChargeConsumption]
PhysicalChannel=DAQ1/ai0
MuxLines=DAQ1/port0/line0,DAQ1/port0
/line1,DAQ1/port0/line2
MuxLatches=ON,OFF,ON
MaxVal=5
MinVal=1
```

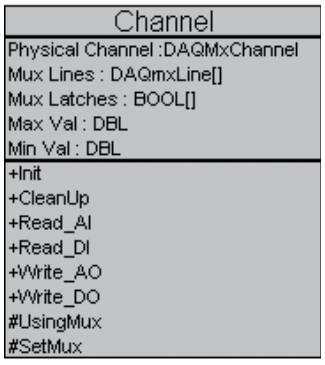

Fig. 7 – Classe Channel aggiornata per gestire il mux

In secondo luogo, il VI nell'UUT che legge l'ini-file deve essere aggiornato. Quindi, la classe Channel viene aggiornata secondo la figura 7. Essa memorizza le linee del mux e le blocca in due array senza alcuna ipotesi circa il numero di linee. La figura 8 illustra la Channel.lvclass:Read_AI.vi aggiornata per la gestione del mux.

L'UsingMux.vi controlla semplicemente se gli array del mux sono vuoti o meno.

Il SetMux.vi applica i latch del mux se il mux viene utilizzato. Questo esempio illustra un beneficio di un progetto orientato agli oggetti: le responsibilità totali dell'applicazione sono distribuite in classi individuali.

In questo caso c'è una classe Channel responsabile della gestione dei canali DAQ.

Il cambiamento del metodo di misura si traduce in un aggiornamento in questa classe, cosa che è naturale.

Esso non ha effetti sul VI dell'esempio di test e ciò è positivo, perché ciò che il test fa non è cambiato, ma è cambiato solo come la misura viene eseguita.

Poiché la classe UUT ha la responsabilità di leggere la configurazione dall'ini-file, anche questo aggiornamento è naturale. Le figure 9 e 10 indicano i VI di test risultanti aggiornati per poter gestire il mux. La complessità della soluzione LabVIEW tradizionale è aumentata, rendendo più difficile la lettura del codice. La soluzione orientata agli oggetti non è invece influenzata dal supporto al mux.

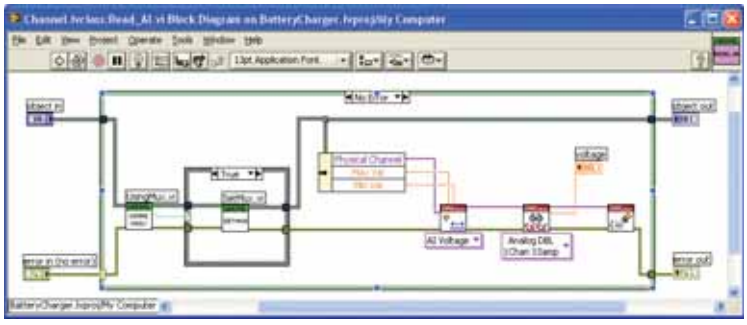

Fig. 8 - Schema a blocchi della Channel.lvclass:Read_AI.vi aggiornata per impostare i latch del mux prima di leggere l'ingresso analogico

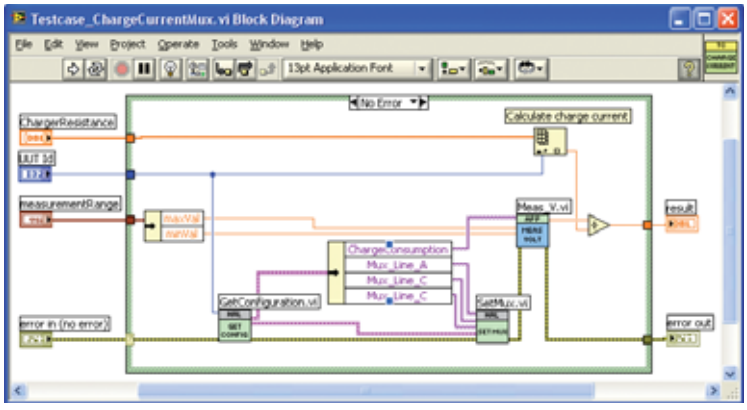

Fig. 9 - Test della corrente di carica, nella soluzione LabVIEW tradizionale, aggiornato per gestire il mux

RIASSUNTO DEGLI EFFETTI DELLA STRUTTURA DEL PROGRAMMA ORIENTATO AGLI OGGETTI

Dopo questa terza modifica del software, la soluzione orientata agli oggetti è più capace e flessibile della soluzione iniziale. E' stata aggiunta funzionalità per consentire al software di gestire l'hardware con o senza mux.

La scelta è configurabile e non si propagherà al VI di test.

Nella soluzione tradizionale, esiste la stessa funzionalità, ma essa influisce sul VI di test, richiedendo quindi una maggiore codifica (quindi, la manutenzione del programma è più costosa).

- Il raggruppamento di dati e VI si traduce in una minore quantità di dati da cablare fra i VI. Ciò rende più semplici la lettura e la manutenzione del codice.

Gli array cablati all'interno dell'applicazione sono spesso sostituiti da una classe con istanze multiple, che incapsulano i dati e le operazioni sui dati.

- Suddividendo l'applicazione in classi, ciascuna con la propria responsabilità, l'applicazione diventa più flessibile e robusta per i cambiamenti.

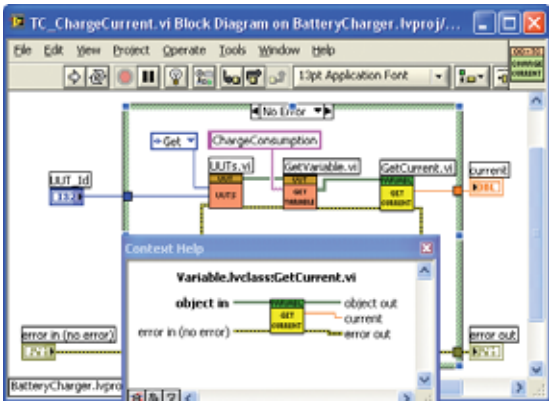

Fig. 10 - Test della corrente di carica, nella soluzione orientata agli oggetti, non influenzata dal supporto al mux

La semplicità con cui è stato aggiunto il mux lo dimostra. Il nuovo requisito non ha provocato alcun cambiamento della struttura del programma.
Anziché propagarsi ai VI all'interno dell'applicazione i cambiamenti provocati dai nuovi requisiti sono spesso localizzati in singole classi. Osservando il progetto nel suo complesso (figura 4), è normalmente molto naturale vedere quale classe deve essere aggiornata per un nuovo requisito.
- La struttura del programma riflette il problema che si sta risolvendo. La classe UUT riflette l'esistenza degli UUT nel

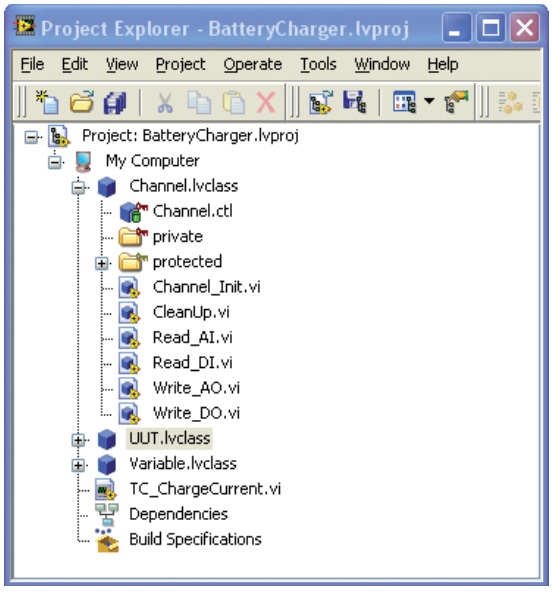

Fig. 11 – Il progetto LabVIEW del Caricabatterie

sistema di test.
Lo stesso vale per la classe Channel. Questo va a vantaggio della comprensione della struttura del programma.
- Infine, c'è un focus maggiore sulla struttura (classi) anzichè sulla funzionalità in termini di VI e subVI.

UNO SGUARDO AL CODICE DELLE CLASSI LABVIEW

La figura 11 descrive il progetto LabVIEW per la soluzione orientata agli oggetti.
Ogni classe è indicata come nodo ClassName.lvclass nel project explorer. Il Classname.ctl viene creato automaticamente per ogni nuova classe LabVIEW per definire i dati della classe. La figura 12 descrive il Channel.ctl che definisce i dati della classe Channel.

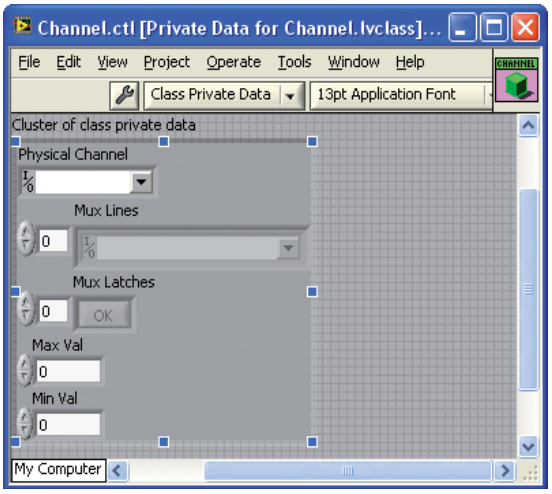

Fig. 12 – Il ctl che definisce i dati della classe Channel

CONCLUSIONE

Nel tempo, tutti i programmi cambiano, spesso più di quanto si preveda. Spesso, inoltre, essi vengono utilizzati in modi non previsti all'inizio del progetto. In questo articolo abbiamo potuto vedere che due programmi molto simili a prima vista, avevano comunque qualità molto differenti in termini di robustezza per i cambiamenti.
Applicare la programmazione orientata agli oggetti può avere effetti positivi sulla struttura del programma, rendendo più facile modificare ed estendere il programma stesso. Ciò permette di fare in modo che la produttività del programmatore non diminuisca al crescere del programma.
Per il programmatore LabVIEW, la programmazione orientata agli oggetti è un modo nuovo di pensare alla struttura del programma e per entrare in questa tecnica può essere necessario un po' di tempo.
Ma ne varrà la pena, sia per i benefici effettivi, sia per il gusto di imparare qualcosa di nuovo.

Note sull'autore

Jan Klasson è Vice President Prodotti & Training presso Endevo – un Alliance Member NI svedese, ed è anche docente di LabVIEW System Design nei corsi Goop in Svezia ed Europa. Egli lavora da 15 anni con lo sviluppo e la metodologia orientati agli oggetti, utilizzando C++, Java e LabVIEW. Jan ha lavorato come consulente utilizzando LabVIEW in diversi progetti di misura, dalla versione LabVIEW 4. Può essere contattato all'indirizzo jan.klasson@endevo.se.