VISUALIZZARE I PROGETTI LABVIEW CON L'USO DELL'UML

Jan Klasson

L'articolo spiega come i programmatori LabVIEW possono trarre vantaggio dall'Unified Modeling Language, o UML, in LabVIEW

Quando si pianificano ed elaborano progetti di sistema, le descrizioni visive sono molto utili. L'UML – Unified Modeling Language – offre una notazione standardizzata per la modellazione grafica dei sistemi. Lo standard ha un'accettazione industriale e sono disponibili tool commerciali. I diagrammi UML possono descrivere sistemi nel loro intero ciclo di vita. Benché molti di questi tipi di diagrammi siano molto utili, il diagramma a classi UML è al primo posto per il progettista. L'articolo spiega che cosa si può descrivere con il diagramma a classi UML, come utilizzarlo e come il supporto dei tool permetta la generazione e la manutenzione automatica dei diagrammi. L'articolo si conclude con una panoramica del diagramma a stati, che è a sua volta un diagramma utile per il programmatore LabVIEW.

PERCHÉ USARE L'UML?

L'UML fornisce i mezzi per comunicare progetti. Questi sono alcuni esempi in cui l'UML può essere utile:

- Descrivere aspetti importanti di una soluzione per discuterla con altri sviluppatori.
- Rivelare il progetto per rendere più semplice la manutenzione.
- Introdurre nuovi programmatori in un progetto.
- Visualizzare le idee e le decisioni di progetto per sé stessi.
- Il supporto dei tool permette di risparmiare tempo quando si crea documentazione.

L'UML ha una vasta accettazione industriale e sono disponibili molti tool per creare diagrammi UML. L'*Endevo GOOP Development Suite – UML Architect Edition* è l'unico tool UML che permette la generazione ed il reverse engineering di codice LabVIEW.

IL DIAGRAMMA A CLASSI UML

La gerarchia dei VI LabVIEW offre una vista ad alto livello di un'applicazione, che indica i singoli VI e le loro dipendenze da altri VI senza alcun dettaglio sul codice. Per presentare il progetto ad un livello più elevato dei singoli VI è necessario un meccanismo di raggruppamento logico. Quando si applica la programmazione orientata agli oggetti, la classe fornisce tale meccanismo di raggruppamento.

Per un'introduzione alla programmazione orientata agli oggetti si può vedere l'articolo pubblicato nel n. 4 di *LabVIEW World*, pubblicato nel 2007. Il diagramma a classi UML indica le classi con caratteristiche e relazioni. Poiché l'UML è indipendente dal linguaggio di programmazione, lo stesso diagramma può essere implementato in qualsiasi linguaggio di programmazione orientata agli oggetti.

La figura 1 illustra un esempio di diagramma a classi UML. Ogni rettangolo rappresenta una classe. Per chiarezza, quando esploreremo il diagramma a classi UML useremo codice LabVIEW.

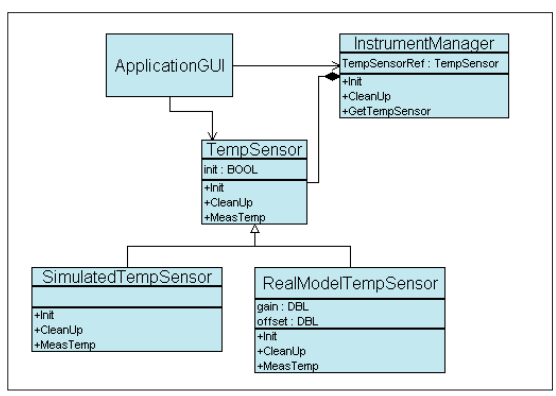


Figura 1 - Diagramma a classi UML

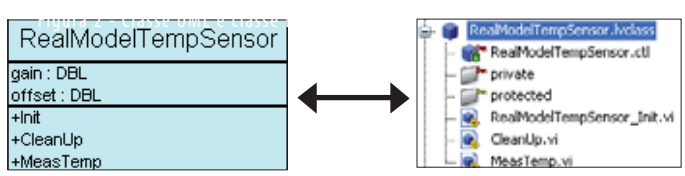


Figura 2 - Classe UML e classe LabVIEW

La figura 2 illustra la classe UML e la lvclass nell'ambiente del progetto. I metodi sono elencati nella parte inferiore della classe UML. In LabVIEW, ogni metodo è realizzato come un VI di metodo. I metodi Init e CleanUp della classe LabVIEW sono responsabili dell'inizializzazione e pulizia dell'oggetto. La sezione intermedia della classe UML elenca gli attributi. Gli

attributi rappresentano l'informazione, o i dati, che è possibile memorizzare in istanze della classe. In LabVIEW, gli attributi sono definiti come campi dati in un typedef (file CTL), ed ogni classe ha la propri type def con lo stesso nome della classe. La figura 3 descrive i campi dati della classe LabVIEW.

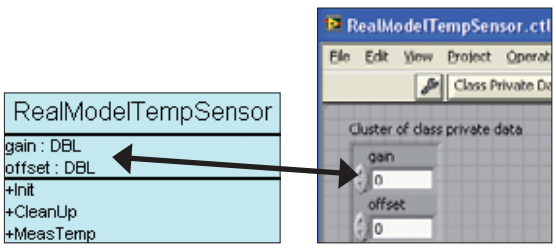


Figura 3 - Attributi UML e dati della classe LabVIEW

CLASSI UML E LABVIEW - RELAZIONI FRA LE CLASSI

L'UML usa varie relazioni per descrivere le dipendenze fra classi. Al contrario, la gerarchia dei VI definisce solo un tipo di relazione, 'usa', per descrivere il fatto che un VI richiama un subVI dal suo diagramma. Per illustrare le relazioni in UML, prenderemo in esame le tre relazioni più utili: eredità, aggregazione composta ed associazione. Altre relazioni sono: dipendenza, aggregazione e realizzazione.

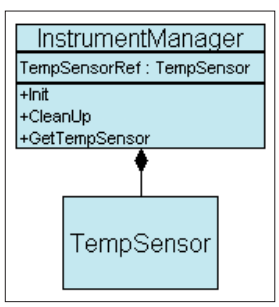


Figura 4 - Aggregazione composta

Le tre classi di sensori di temperatura della figura 1 sono incluse in una gerarchia di eredità. Questa relazione ha una corrispondenza diretta in LabVIEW.
Le classi LabVIEW ereditano fra loro esattamente nello stesso modo.

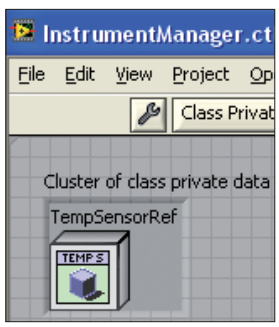


Figura 5 - Controllo della classe TempSensor nei dati InstrumentManager

Nell'UML, l'aggregazione composta è indicata da una linea con un rombo nero, come si vede nella figura 4. Questa relazione significa che la classe InstrumentManager è una repository per gli oggetti TempSensor.
Logicamente, gli oggetti TempSensor sono memorizzati all'interno di oggetti InstrumentManager.
In LabVIEW, il controllo della classe TempSensor è incluso nei dati oggetto di InstrumentManager, come si vede nella figura 5.

La figura 6 indica come un oggetto inizializzato della classe RealModelTempSensor è creato e memorizzato all'interno dl metodo Init della classe InstrumentManager.

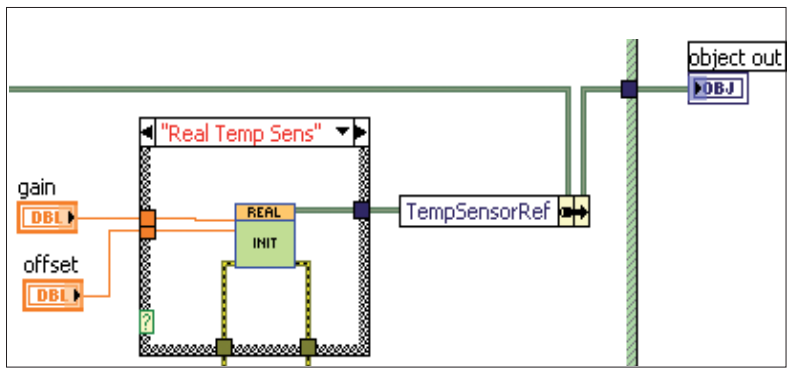


Figura 6 - Oggetto RealTempSensor creato e memorizzato da InstrumentManager

L'aggregazione composta è un utile modello di progettazione, perché definisce un contenitore per gli oggetti. Per ottenere un oggetto sensore di temperatura inizializzato, il codice client, per esempio ApplicationGUI, deve semplicemente

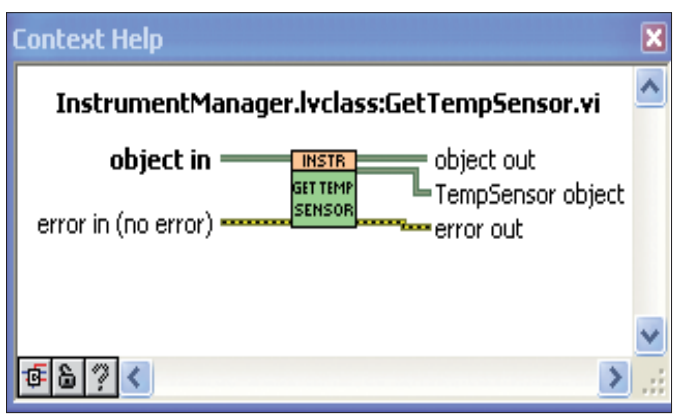


Figura 7 - Il metodo GetTempSensor della classe InstrumentManager

chiamare il VI del metodo GetTempSensor (figura 7) della classe InstrumentManager.
Un'altra relazione utile è l'associazione.
Essa è più vaga dell'aggregazione composta ed è spesso utilizzata per indicare una relazione 'usa', come illustrato nella figura 8.

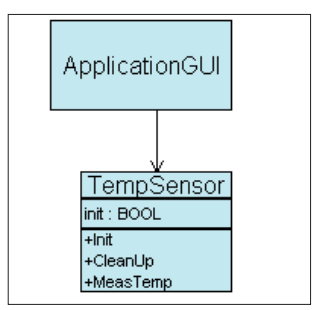


Figura 8 - La relazione di associazione

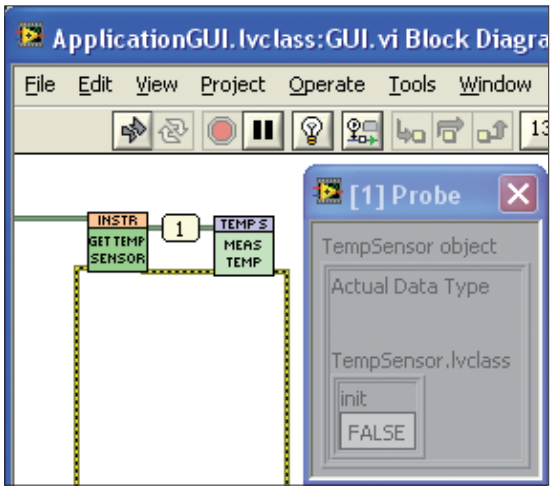


Figura 9 - Uso dell'oggetto TempSensor

La figura 9 descrive il codice all'interno del metodo GUI di ApplicationGUI che causa la dipendenza visualizzata come relazione di associazione.

USO DEL DIAGRAMMA A CLASSI UML

Il diagramma a classi è utile al progettista quando deve creare la bozza di una soluzione o descrivere il progetto completo.

Poiché lo scopo principale del diagramma è quello di creare una comprensione del progetto, è inutile aggiungere informazioni dettagliate se queste non migliorano la comprensione stessa. Aggiungere classi e relazioni è un punto d'inizio naturale per il progetto. Anche i metodi pubblici sono importanti, perché forniscono informazioni sui servizi messi a disposizione da una classe.
Il termine 'pubblico' denota la visibilità dei metodi. *Pubblici* significa che possono essere utilizzati da metodi di altre classi. *Privati* significa che possono essere utilizzati solo dall'interno della classe. *Protetti* significa che sono condivisi fra classi in una gerarchia di eredità. Normalmente, i metodi protetto e privato sono caratteristiche che è meglio aggiungere durante la codifica.

I metodi possono avere parametri in UML – controlli e indicatori in LabVIEW – ma normalmente i parametri non aggiungono nulla alla comprensione del progetto, quindi è preferibile aggiungerli nel codice.

Gli attributi sono meno importanti dei metodi e non vengono normalmente aggiunti durante la progettazione, a meno che il diagramma sia generato da tool. Infatti, una classe è prevalentemente definita dai servizi che fornisce (mediante metodi pubblici); le capacità di memorizzazione dei dati sono aggiunte quando vengono implementati i metodi.

Le classi sono fornitori di servizi, non un modo per implementare dati globali.
Tuttavia, sono un meccanismo eccellente per evitare dati globali.

TOOL DI SUPPORTO PER MANTENERE AGGIORNATI I DIAGRAMMI

Molti tool UML forniscono moduli di connessione del codice. L'*Endevo GOOP Development Suite*, usato per creare tutti i diagrammi in questo articolo, offre una connessione di codice per le classi LabVIEW. I tool di connessione del codice possono fornire queste caratteristiche per i diagrammi a classi UML (l'*Endevo GOOP Development Suite* supporta tutti e tre):

- **Generazione codice** – Supporta la generazione di codice da un diagramma a classi.
- **Reverse Engineering** – Supporta la generazione di diagrammi a classi da codice esistente.
- **Sincronizzazione** – Supporta la manutenzione dei diagrammi quando il codice o il diagramma cambiano.

Probabilmente, la caratteristica più importante è il reverse engineering, perché permette di creare la corretta documentazione del progetto – il diagramma a classi – in qualsiasi punto dello sviluppo. Nella generazione del codice si risparmia tempo, perché non è necessario creare manualmente tutti gli elementi di codice descritti nel diagramma a classi, tuttavia rimane un'operazione eseguita una sola volta durante lo sviluppo.

La sincronizzazione, insieme al reverse engineering, offre un modo di lavoro nel quale il progetto è 'sempre' aggiornato. Ogni volta che è necessario un progetto corretto, si può eseguire la sincronizzazione. Se è supportato solo il reverse engineering, il diagramma non sarà aggiornato altrettanto spesso. La ragione è che dopo ogni operazione di reverse engineering è necessario aggiornare manualmente il layout grafico (è difficile che i tool possano fornire un layout perfetto). Non si tratta di molto lavoro, ma occorre qualche minuto.

ANALISI DEL CODICE E CONVENZIONI DI MODELLAZIONE

Gli aggiornamenti dell'UML supportati da tool non sono solo più efficaci in termini di tempo rispetto alla manutenzione manuale dei diagrammi. Il tool fornisce effettivamente l'analisi automatizzata del codice, promettendo perciò una maggiore correttezza rispetto ai diagrammi aggiornati manualmente. L'analisi mediante tool può trovare elementi facilmente trascurati dallo sviluppatore.

I diagrammi generati da tool sono anche più coerenti per quanto riguarda le relazioni fra le classi, perché vengono

applicate ogni volta le stesse regole. Quando crea manualmente i diagrammi, il progettista ha flessibilità nella scelta delle relazioni. Le definizioni di relazioni dell'UML non sono così rigorose da rendere possibile un solo modo per fare 'correttamente'. Per rendere i diagrammi UML facilmente comprensibili all'interno di un gruppo di sviluppatori, è necessario applicare qualche convenzione sull'uso delle relazioni. I tool di reverse engineering forniscono tale convenzione.

REVERSE ENGINEERING E SINCRONIZZAZIONE

La figura 10 descrive il dialogo di reverse engineering che viene lanciato dall'interno del progetto. Tutte le classi sono aggiunte automaticamente al dialogo. In alternativa, è possibile selezionare singole classi. La figura 11 mostra il diagramma a classi risultante; le classi sono state spostate per migliorare il layout, ma tutte le informazioni sono state generate da tool. Quando una classe UML è stata collegata al codice (automaticamente durante il reverse engineering), sulla classe sono disponibili le operazioni di sincronizzazione. Nell'UML si possono aggiungere metodi ed attributi (campi dati) e i cambiamenti possono essere sincronizzati al codice e viceversa.

USO DEL DIAGRAMMA A CLASSI PER CODICE NON ORIENTATO AGLI OGGETTI

L'ApplicationGUI è nel nostro esempio un lvclass con un VI metodo GUI. Se fosse stato semplicemente un VI - ApplicationGUI.vi – avrebbe potuto ancora essere visualizzato in UML. In questo caso, la classe UML avrebbe utilizzato un'annotazione, <<GUI>>, prima del nome della classe (figura 11). Tale annotazione viene chiamata tipo stereo in UML ed è usata per aggiungere semantica alla classe UML. In questo caso, <<GUI>> denota che si utilizza il simbolo della classe per il VI GUI dell'applicazione, che non è una classe. Sembra naturale avere l'effettivo VI GUI nel diagramma a classi, nonostante non sia una classe. Esso contribuisce, infatti, alla comprensione del progetto.

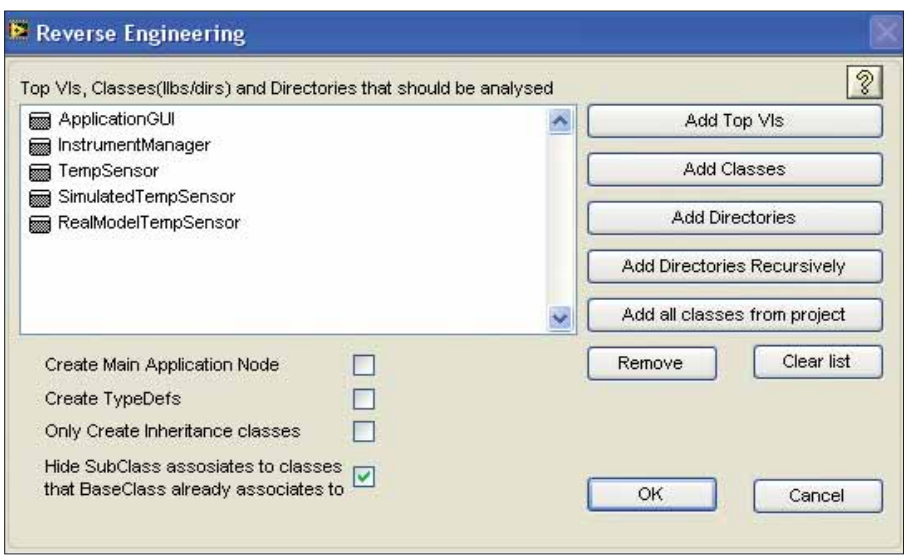
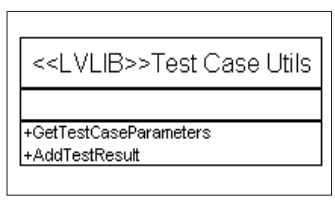


Figura 10 - Dialogo del reverse engineering

Figura 11 - Illustrazione di un VI GUI come classe UML con il tipo stereo 'GUI'



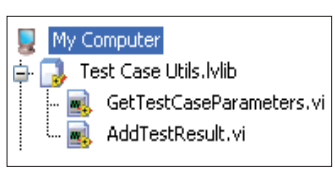


Figura 12 - VI di un lvlib visualizzati in UML

Un altro modo di usare il simbolo della classe UML per codice non orientato agli oggetti è il raggruppamento di VI.
La figura 12 mostra i VI di un lvlib visualizzati come classe UML usando il tipo stereo 'LVLIB'.
Il nome del lvlib è usato come nome della classe in UML.

Se non si usano i lvlib, come nome della classe si usa il concetto comune rappresentato dai VI, come i VI Binary File. In questo caso, il tipo stereo sarebbe 'VI'. I VI del raggruppamento logico sono descritti come metodi in UML.

L'applicazione di questa tecnica rende possibile usare l'UML anche se non viene utilizzata la programmazione orientata agli oggetti. Rispetto alla gerarchia dei VI, questa tecnica mette in luce una responsabilità comune fra i gruppi di

VI, mentre la gerarchia dei VI si focalizza solo su relazioni 'usa' fra i singoli VI. Questa tecnica rende inoltre possibile elaborare le dipendenze fra i gruppi di VI usando le relazioni UML.

ALTRI DIAGRAMMI UML

Vi sono molti aspetti differenti della progettazione di sistemi e non è possibile catturarli tutti in un solo tipo di immagine. L'UML definisce quindi una serie di tipi di diagrammi, dove ogni tipo cattura determinati aspetti del progetto. Oltre al diagramma a classi, un altro diagramma utile al programmatore LabVIEW è il diagramma a stati. Descriveremo quindi anche questo tipo di diagramma.

IL DIAGRAMMA A STATI

Il diagramma a stati è di grande interesse per il programmatore LabVIEW perché spesso nei VI vengono implementate macchine a stati o handler di messaggi in coda.
È, per esempio, molto comune implementare VI UI come VI di macchine a stati.
Le macchine a stati sono un potente meccanismo di implementazione, ma può essere difficile ottenere una visione d'insieme di tutte le possibili transizioni di stato dal codice.
L'*Endevo GOOP Development Suite* può generare diagrammi a stati da un VI di macchina a stati. Le figure 13 e 14 lo illustrano usando un VI creato come esempio il template della macchina a stati standard LabVIEW.
NI fornisce anche il toolkit del diagramma a stati, che permette al programmatore di creare diagrammi a stati ed ottenere per essi il codice LabVIEW.

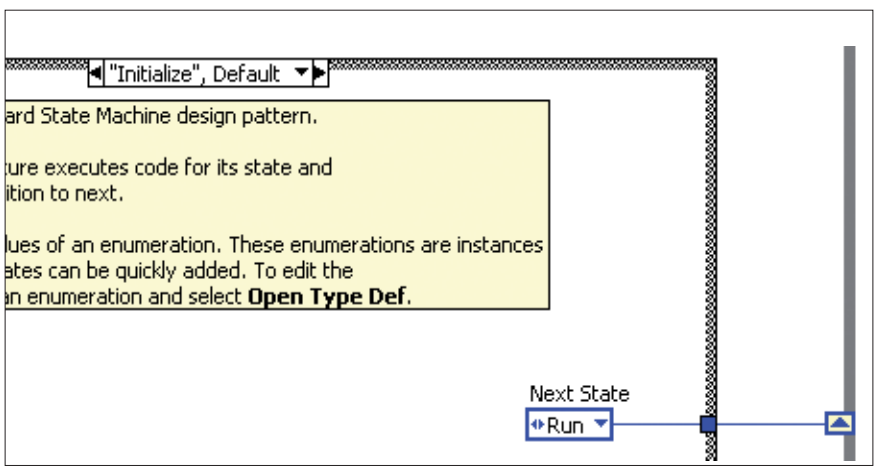


Figura 13 - Parte dello schema a blocchi da una macchina a stati LabVIEW

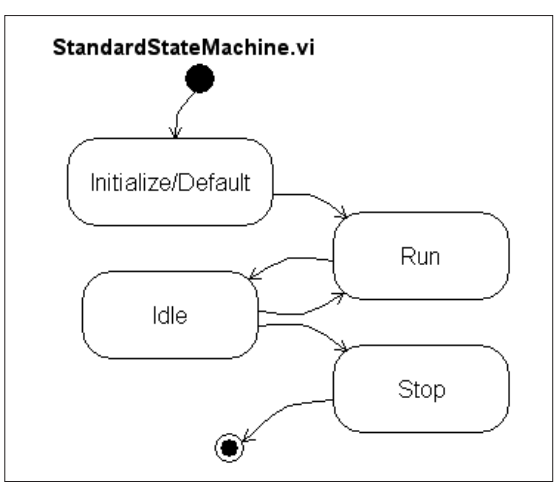


Figura 14 - Diagramma a stati UML

CONCLUSIONE

L'articolo ha esplorato come si possono usare i diagrammi a classi e a stati UML per visualizzare il progetto di programmi LabVIEW. Per essere produttivi durante la progettazione, è vitale un buon supporto di tool.
Abbiamo illustrato come si può usare l'*Endevo GOOP Development Suite* non solo per la modellazione, ma anche per la generazione automatizzata della documentazione del progetto.
Usando la funzionalità di collegamento del codice, l'*Endevo GOOP Development Suite* diventa un tool per esplorare visivamente il codice da un punto di vista del progetto.
Per ulteriori informazioni sull'UML, visitate *www.uml.org*. Visitate anche *www.endevo.se* per altri documenti e demo video. Un eccellente libro sull'UML è UML Distilled (3a edizione) di Martin Fowler (ISBN 0-321-19368-7).

Note sull'autore

Jan Klasson è Vice President of Products & Training presso Endevo – un Alliance Partner di NI svedese – e docente di un corso su LabVIEW Object Oriented System Design in Svezia ed Europa. Lavora da 15 anni su sviluppo e metodologie orientati agli oggetti usando C++, Java e LabVIEW. Jan ha inoltre operato come consulente utilizzando LabVIEW in diversi progetti di misura da LabVIEW 4. Può essere contattato all'indirizzo: jan.klasson@endevo.se.