Lo standard IEC 61499 per sistemi distribuiti di automazione industriale

Luca Ferrarini, Carlo Veber

Nell'articolo sarà introdotto il problema dell'implementazione di un modello di esecuzione di un'applicazione IEC 61499. In questa normativa, un'applicazione è definita come un network di blocchi funzionali i cui nodi sono i blocchi funzionali stessi e i cui rami sono costituiti dalle connessioni di dati ed eventi. La normativa specifica anche il modello di esecuzione di un blocco funzionale per mezzo degli stati macchina e dei diagrammi temporali. Queste specifiche non sono esaurienti ai fini dell'implementazione di un tale modello e consentono differenti soluzioni, tutte compatibili con le specifiche, ma in grado di produrre comportamenti diversi. In questo lavoro alcune di queste differenti soluzioni sono presentate, implementate e provate con l'intento di sottolineare i vantaggi e gli svantaggi di ciascun approccio. Infine, è presentato un esempio di applicazione per il controllo del moto di un giunto robotico, sia localmente che in un sistema distribuito.

Keyword
Sistemi Operativi Multitasking, Standard IEC 61499, Tecniche Object-Oriented

Nell'ultimo ventennio il campo del controllo di sistemi d'automazione industriale ha visto una crescente innovazione tecnologica sia per ciò che concerne gli strumenti hardware, si pensi ad esempio alla diffusione di bus di campo con prestazioni sempre più elevate come FieldBus, Profibus, Can, sia per ciò che riguarda gli strumenti software, con particolare riferimento alla crescente diffusione e utilizzo dei linguaggi della IEC 61131-3 [1]. In questi ultimi anni la International Electrotechnical Commission ha attivato un progetto che sta portando alla definizione di uno standard per il progetto e la realizzazione delle funzioni di comando per sistemi distribuiti di controllo e misura di processi industriali (Industrial-Process Measurement and Control System, Impcs). Questo standard, chiamato IEC 61499 [3, 5, 7], già seguito e analizzato in ambito accademico [2, 3, 8], comincia oggi a suscitare l'interesse anche del mondo industriale.

Le applicazioni costruite con questo standard sono basate su un elemento fondamentale: il blocco funzionale. Il blocco funzionale è un'entità funzionale software associata ad una risorsa hardware del sistema di controllo. Un'applicazione è composta da un insieme di blocchi funzionali interconnessi tra loro tramite connessioni eventi e connessioni dati che ne determinano il flusso di esecuzione. Al contrario di quanto accade per i blocchi funzionali definiti nella IEC 61131-3, questi blocchi hanno uno stretto legame con il concetto di oggetto presente nell'ambito informatico. Esso possiede un gestore, chiamato Execution Control (EC), che sulla base degli eventi che il blocco riceve in entrata, dei dati di ingresso e sullo stato del blocco, esegue delle operazioni e genera eventi di uscita. Nel caso particolare dei blocchi funzionali di base, questo gestore viene chiamato EC Chart (ECC), ed è definito come una macchina a stati in cui allo stato possono essere associate delle azioni (composte da algoritmi da eseguire e da eventi di uscita da generare) mentre l'arco rappresenta una transizione a cui sono associate delle condizioni che, se verificate, permettono il passaggio da uno stato al successivo. Lo standard propone una specifica per l'esecuzione di un blocco funzionale e di un'applicazione descritta da due macchine a stati e da un diagramma temporale. Sfortunatamente, questa specifica risulta non esaustiva nel caso in cui si voglia implementare uno strumento per lo sviluppo dei modelli IEC 61499; essa infatti permette l'implementazione di soluzioni differenti, come mostrato anche in [9], tutte conformi alla specifica, ma che realizzano comportamenti diversi tra loro. Il problema affrontato qui sostanzialmente in modo locale, cioè per un'applicazione istanziata su una singola risorsa, si ripercuote anche su applicazioni distribuite su dispositivi diversi, come mostrato in [6].

L'articolo è strutturato come segue. Nel prossimo paragrafo verranno introdotti e analizzati i modelli che governano l'evoluzione di un'applicazione e in particolare di un blocco funzionale. Successivamente, nel terzo paragrafo vengono presentati dei possibili approcci implementativi basati sulle tecniche object-oriented. Nel quarto paragrafo verranno scelti, implementati e analizzati quattro approcci tra quelli definiti nel paragrafo precedente sottolineando di ognuno vantaggi e svantaggi. Infine nell'ultimo paragrafo viene presentato come esempio di applicazione IEC 61499 il controllo del moto di un giunto robotico, non solo eseguito localmente ma anche distribuito su dispositivi diversi.

L. Ferrarini, C. Veber - Dipartimento di Elettronica e Informazione, Politecnico di Milano

Modelli che governano l'evoluzione di un'applicazione IEC 61499

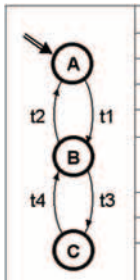
La normativa IEC 61499 specifica il modello di esecuzione di un blocco funzionale definendo gli istanti temporali in cui il blocco interagisce con la risorsa e con il resto dell'applicazione. Queste interazioni, mostrate nella figura 1, sono qui di seguito riportate:

- t1: i dati rilevanti per un evento sono resi disponibili e stabili all'ingresso del blocco funzionale;
- t2: occorrenza di un evento all'interfaccia eventi;
- t3: la funzione di controllo dell'esecuzione ECF richiede alla funzione di schedulazione di eseguire l'algoritmo associato allo stato attuale dell'ECC;
- t4: la funzione di schedulazione rende disponibile la risorsa e inizia l'esecuzione dell'algoritmo;
- t5: l'algoritmo completa il calcolo delle variabili di uscita, e le rende disponibili sulle uscite per i dati;
- t6: il blocco funzionale comunica alla funzione di schedulazione che ha terminato l'esecuzione dell'algoritmo e rilascia la risorsa;
- t7: la funzione di schedulazione comunica all'ECF che è terminata l'esecuzione dell'algoritmo;
- t8: l'ECF genera gli eventi in uscita associati allo stato appena terminato rendendoli disponibili sulle uscite dell'interfaccia eventi.

Sulla base di questi istanti temporali la normativa definisce quattro intervalli:

- T_{setup} = t2 - t1;
- T_{start} = t4 - t2 (tempo tra l'occorrenza di un evento all'interfaccia di ingresso e l'inizio dell'esecuzione del primo algoritmo);
- T_{alg} = t6 - t4 (tempo di esecuzione di un algoritmo);
- T_{finish} = t8 - t6 (tempo tra la fine dell'esecuzione dell'algoritmo e la generazione dell'evento di uscita).

Il comportamento di un'istanza di un blocco funzionale è formalizzato attraverso due tipologie di macchine a stati finiti. La prima specifica il comportamento del blocco all'occorrenza di uno specifico evento. La macchina a stati in stato di *pronto* controlla l'interfaccia degli eventi di ingresso fino all'occorrenza dell'evento. Quando l'evento viene registrato viene fatta richiesta di attivare l'ECC e quando questa viene servita la macchina a stati che controlla l'evento va in uno stato di *attesa* fino al completamento di un *passo* dell'ECC (il significato del termine passo sarà chiarito nel seguito), dopodiché ritorna nello stato di pronto. La seconda macchina a stati, descritta in tabella 1, regola l'esecuzione dell'ECC. Nel momento in cui l'ECC viene attivato lo stato attivo diventa lo stato B. Se una transizione dell'ECC può essere superata gli algoritmi associati allo stato successivo vengono schedulati e poi eseguiti. Durante l'esecuzione degli algoritmi la macchina a stati è nello stato C. Al termine dell'esecuzione degli algoritmi vengono settati i valori delle variabili corrispondenti agli eventi di ingresso (posti a 0) e agli eventi di uscita da generare (posti a 1) e viene aggiornato lo stato dell'ECC. A questo punto se ci sono altre transizioni che possono scattare vengono schedulati nuovamente gli algoritmi (se presenti) dello stato a valle (transizione t3) altrimenti vengono generati tutti gli eventi di uscita e la macchina a stati dell'esecuzione dell'ECC torna nello stato A (nel seguito questo ciclo verrà indicato con il termine "passo").



Stato	Condizione	
A	pronto	
B	schedulazione degli algoritmi	
C	attesa del completamento degli algoritmi	
Transizione	**Condizione**	**Operazione**
t1	invocazione ECC	impostazione delle variabili EI conferma valore ingressi valutazione delle transizioni
t2	nessuna transizione attiva	generazione degli eventi
t3	una transizione attiva	inizio esecuzione algoritmi
t4	completamento degli algoritmi	cancellazione delle variabili EI impostazione delle variabili EO valutazione delle transizioni

Tabella 1 - Macchina a stati dell'esecuzione dell'ECC

La macchina a stati della tabella 1 non è un modello temporale in quanto il tempo non è esplicitamente rappresentato.
In una macchina a stati esistono due tipologie di intervalli temporali: il tempo di permanenza in uno stato e il tempo di esecuzione della transizione da uno stato ad un altro.
È evidente però, la volontà di rendere questo secondo intervallo temporale il più piccolo possibile, al limite nullo.
La normativa affronta il problema nella descrizione dei blocchi speciali (Annesso A, Parte I), dicendo the i tempi T_{setup}, T_{start} e T_{finish}, la cui mappatura nella macchina a stati della tabella 1 è evidente), sono da considerarsi uguali a 0. È altresì ovvio che in fase di implementazione questa ipotesi non è attuabile.
La mancanza di vincoli temporali porta, come si vedrà in seguito, a diverse implementazioni dei modelli della normativa IEC 61499, tutti conformi alla specifica, ma che realizzano in realtà comportamenti diversi tra loro.

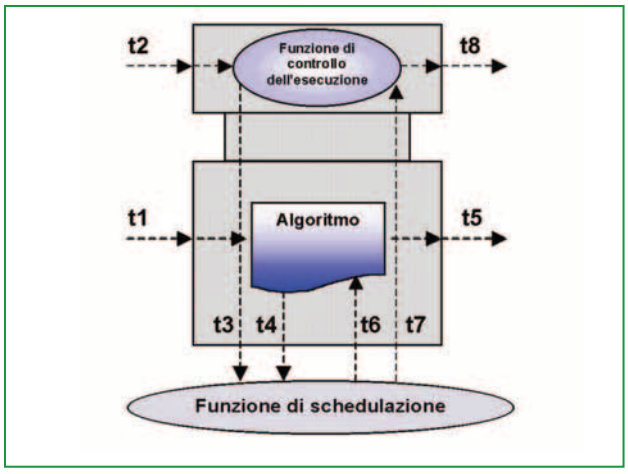


Figura 1 - Interazione del blocco funzionale con la risorsa

Possibili approcci implementativi

I possibili approcci implementativi, basati su un modello ad

oggetti derivato da quello presente nella normativa, possono essere raggruppati secondo i criteri di: scansione ordinata dei blocchi funzionali; implementazione multitasking.
Nel caso della scansione ordinata è possibile gestire i blocchi funzionali secondo due metodologie opposte. In un caso la scansione ordinata è fissata a priori, similmente a quanto accade per un sistema basato su PLC. Nell'altro caso, la scansione ordinata non è fissata a priori, ma dipende solo dall'occorrenza degli eventi. Queste due implementazioni vengono spesso definite come *time-driven* la prima, ed *event-driven* la seconda [6].
Nel caso di implementazione multitasking è possibile fare una prima distinzione sull'utilizzo o meno delle caratteristiche di questi sistemi. Nel caso in cui il multitasking venga utilizzato, è possibile suddividere ulteriormente gli approcci sulla base del fatto che utilizzino una politica di sospensione o meno dei processi associati ai blocchi funzionali o basata semplicemente sull'esecuzione di un singolo passo dell'ECC. Le possibili combinazioni sono rappresentate sinteticamente in tabella 2 e spiegate nel seguito.

Nota – la "×" in tabella indica la non implementabilità del metodo.

Multitasking / Scansione Ordinata	Non usato	Usato con nessuna politica di sospensione	Usato con politica di sospensione, timeslice	Usato con esecuzione di un passo dell'ECC
Non fissata	a_0	a_1	a_2	a_3
Fissata	a_4	×	a_5	a_6

Tabella 2 - Possibili metodi d'implementazione

- a_0: il blocco funzionale è implementato come oggetto semplice e il passaggio di un evento è implementato attraverso una chiamata funzionale diretta al metodo che implementa il codice dell'ECC del blocco funzionale successivo;
- a_1: il blocco funzionale è implementato come un singolo processo. Tutti i processi sono eseguiti in parallelo e la segnalazione di un evento è ottenuta tramite opportune tecniche di sincronizzazione;
- a_2: simile al metodo A1, in questo caso però i processi vengono schedulati con una politica a timeslice;
- a_3: il blocco funzionale è implementato come un singolo processo; ogni processo rimane attivo fino a quando la macchina a stati che governa l'ECC non torna in stato di *pronto*;
- a_4: il blocco funzionale è implementato come oggetto semplice; l'esecuzione è ottenuta tramite un algoritmo che emula il funzionamento di un PLC, il quale, seguendo un ordine prestabilito dell'esecuzione dei blocchi, chiama direttamente il metodo di esecuzione dell'ECC del blocco da eseguire;
- a_5: il blocco è implementato come un singolo processo; l'esecuzione è ottenuta tramite un algoritmo che emula il funzionamento di un PLC, il quale, seguendo un ordine prestabilito dell'esecuzione dei blocchi, permette l'esecuzione di un solo processo per un tempo prestabilito ΔT, tenendo gli altri processi in uno stato di sospensione.
- a_6: il blocco è implementato come un singolo processo; l'esecuzione è ottenuta tramite un algoritmo che emula il funzionamento di un PLC, il quale, seguendo un ordine prestabilito dell'esecuzione dei blocchi, permette l'esecuzione di un solo processo fino a quando questo non ha completato un passo dell'ECC (fino a quando la macchina a stati dell'esecuzione dell'ECC ritorna nello stato di *pronto*), tenendo gli altri processi in uno stato di sospensione.

L'ambiente di sviluppo più conosciuto (e gratuito) che permette di editare e simulare l'architettura definita dalla normativa IEC 61499 è sicuramente il Function Block Development Kit (Fbdk) di Rockwell Automation [4].
Questo software permette inoltre la generazione automatica di codice Java eseguibile poi su una qualunque Java Virtual Machine (JVM). L'ultima versione rilasciata (settembre 2003) permette di sviluppare applicazioni seguendo l'approccio a_0. Questo approccio pur rispettando le specifiche della normativa presenta alcuni comportamenti indesiderati come ad esempio quelli mostrati nella figura 2. Con riferimento alla figura 2a, si supponga che l'ECC del blocco FB1 si trovi nello stato S1 pronto a generare l'evento di uscita EO (a), e che EO sia connesso con un evento di ingresso di FB2, che a sua volta sia connesso a FB3. Si supponga inoltre che la condizione associata alla transizione a valle di S1 abbia valore 1 (d). In questo caso lo scatto della transizione può avvenire solamente dopo le chiamate (b) e (c) e il loro ritorno. In modo analogo, nel caso della figura 2b, la generazione dell'evento sulla connessione (d), cioè la chiamata diretta tra FB1 e FB4 può avvenire solo dopo la chiamata tra FB1 e FB2 (b), tra FB2 e FB3 (c) e il ritorno di queste chiamate. In questo modo il comportamento di un blocco funzionale all'interno di un'applicazione viene a dipendere fortemente dal numero di blocchi funzionali presenti e dalla topologia della rete, rendendo di fatto il comportamento di un blocco funzionale non predicibile.

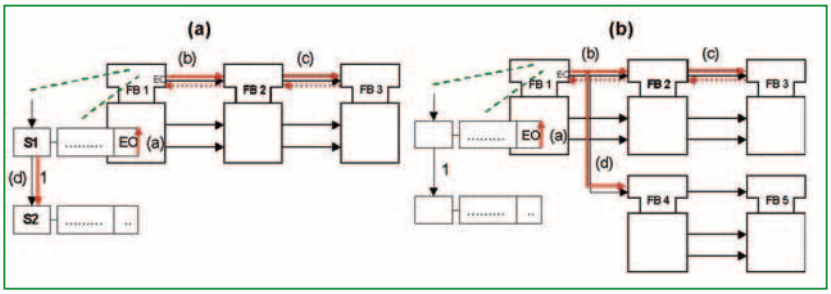


Figura 2 - Esempi che impediscono la predicibilità dell'esecuzione di un ECC

Fbdk mette però a disposizione una libreria di blocchi precompilati in grado di "spezzare la catena", in grado cioè di far ritornare immediatamente il controllo al chiamante. Ad esempio, nel caso di figura 2b, se FB2 fosse un blocco della libreria Fbdk con queste caratteristiche, allora alla generazione dell'evento EO da parte di FB1 corrisponderebbe una chiamata diretta a FB2 (b) ma questa ritornerebbe subito,

permettendo "subito" la chiamata verso FB4. In questo modo viene però imposto al progettista di conoscere i dettagli implementativi e di dover porre rimedio a questi problemi o inserendo dei blocchi funzionali speciali o modificando lui stesso il codice di controllo prodotto.

Implementazione in Java e analisi

Degli approcci descritti in tabella 2, ne sono stati selezionati 4 che mostrano comportamenti diversi e molto interessanti, essi sono a_0, a_3, a_4 e a_5. Per le loro peculiarità si è deciso di dare a questi approcci un nome ben preciso: Direct Call (a_0), Thread (a_3), PLC (a_4) e Dispatcher (a_5).
Nel seguito verrà presentata una breve descrizione dell'implementazione dei suddetti approcci utilizzando come linguaggio di programmazione Java.

Direct Call
Ogni blocco funzionale è implementato come oggetto distinto, mentre le connessioni eventi sono gestite da un oggetto unico. In un oggetto di tipo blocco funzionale, le variabili interne del blocco sono gli attributi dell'oggetto, gli algoritmi sono i metodi, mentre il comportamento dell'ECC è implementato nel metodo *run()* dell'oggetto. Tutti gli oggetti che rappresentano un blocco funzionale possiedono inoltre un metodo *initialize()* per l'inizializzazione. L'oggetto che gestisce la connessione eventi contiene un metodo *setEvent(event)* che sequenzialmente chiama il metodo *run()* degli oggetti corrispondenti ai blocchi funzionali in attesa di *event*. La generazione di un evento è quindi implementata nel metodo *run()* dell'oggetto con una chiamata diretta al metodo *setEvent(event)* dell'oggetto che rappresenta la connessione eventi.

Thread
Ogni blocco funzionale è implementato con un oggetto che estende la classe *Thread* (*Runnable*). Questo oggetto contiene gli stessi attributi e metodi dell'oggetto blocco funzionale dell'approccio Direct Call ma ha in più un attributo *EventBuffer* e alcuni metodi per gestirlo. L'oggetto *EventBuffer* viene istanziato nel metodo *initialize()* e implementa la sua interfaccia eventi di ingresso. Il metodo *setEvent(event)* della classe *EventBuffer* aggiunge l'evento in una lista degli eventi accaduti ed esegue la funzione *NotifyAll()*. Le connessioni evento sono anch'esse implementate come nel primo approccio ma il metodo *setEvent(event)* chiama il metodo *setEvent(event)* dei corrispondenti *EventBuffer*. L'implementazione dell'*EventBuffer* permette di far terminare questa chiamata in un tempo molto breve e prefissato.
Quando un ECC è nello stato di *pronto* (con riferimento alla macchina a stati della tabella 1) il blocco viene risvegliato da una *NotifyAll()*, quindi legge sequenzialmente la lista degli eventi accaduti e non ancora serviti all'interno del *EventBuffer* finché non ne trova uno che permette di superare una transizione dell'ECC. Se ciò non accade, l'ECC sospende la sua esecuzione tramite una *wait()* e ritorna in uno stato di pronto (con riferimento alla macchina a stati di tabella 1). Altrimenti viene eseguito un passo dell'ECC, attraverso le transizioni t3 e t4 della macchina a stati di tabella 1, fino al ritorno in uno stato di *pronto*.

PLC
Ogni blocco funzionale è implementato con un oggetto così come ogni connessione eventi. Un oggetto connessione eventi contiene un attributo per il suo ingresso (l'evento di uscita del blocco a monte) e un certo numero di attributi corrispondenti alle sue uscite (gli eventi di ingresso dei blocchi a valle). Questo oggetto possiede dei metodi per gestire questi attributi e un metodo *run()* che esegue l'assegnazione del valore dell'attributo di ingresso a tutti gli attributi di uscita. L'oggetto che rappresenta il blocco funzionale è implementato come nell'approccio Direct Call, ma il suo metodo *run()* permette di eseguire un passo dell'ECC ad ogni chiamata. L'esecuzione della rete di blocchi funzionali è determinata da un oggetto che emula il comportamento di un PLC. Esso esegue ciclicamente la sequenza di metodi *run()* dei blocchi funzionali presenti nell'applicazione e successivamente aggiorna tutte le connessioni (eseguendo il loro metodo *run()*).

Dispatcher
Questo approccio è simile al precedente ma in questo caso gli oggetti sono implementati come thread. In questo modo è possibile modificare il metodo *run()* in modo tale che l'oggetto PLC possa far eseguire il blocco funzionale non per un passo dell'ECC ma per un tempo prestabilito ΔT.

Analisi
Gli approcci qui presentati presentano vantaggi e svantaggi. La loro analisi è stata fatta tramite numerosi esperimenti. Qui nel seguito riportiamo i più significativi.
Il confronto è stato fatto sulla base degli istanti temporali definiti nella normativa e riportati nel primo capitolo. Le particolari implementazioni hanno portato all'esigenza di inserire un nuovo intervallo temporale T_{wake}, rappresentato nella figura 3, che indica l'intervallo tra la mappatura degli ingressi ed il momento in cui il blocco funzionale è realmente schedulato sulla risorsa.

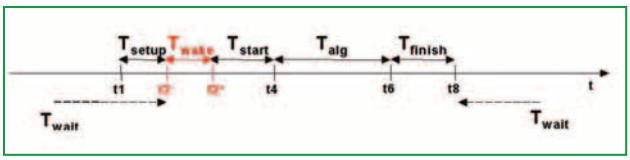


Figura 3 - Intervalli temporali principali

Gli esperimenti sono stati fatti utilizzando, oltre al blocco funzionale *E_RESTART*, un unico blocco funzionale chiamato *LOOP* rappresentato in tabella 3. Esso ha un evento di ingresso *REQ* e un evento di uscita *CNF*. All'occorrenza dell'evento di *REQ* il blocco esegue l'algoritmo REQ (tabella 3c) e alla sua terminazione genera l'evento *CNF*, secondo l'ECC della tabella 3b.

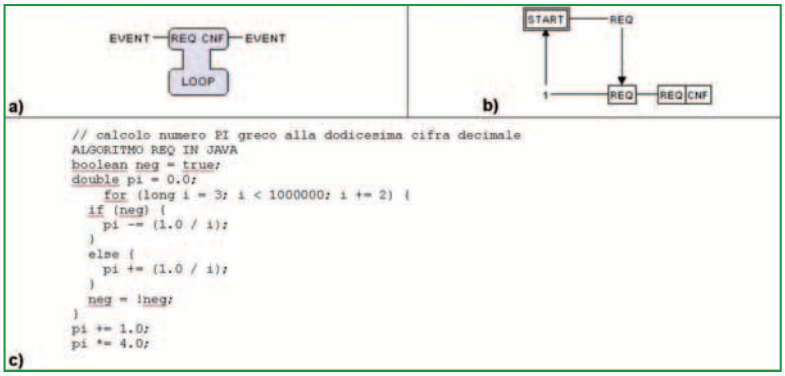


Tabella 3 - Il blocco funzionale campione: interfaccia (a), ECC (b), algoritmo REQ (c)

Il *primo esperimento* ha l'obiettivo di stabilire il tempo di esecuzione del singolo blocco quando questo è l'unico elemento dell'applicazione. In particolare come tempo di esecuzione si intende l'intervallo tra t2+ e t8. I risultati di questo esperimento sono mostrati in tabella 4, dove il numero associato al Dispatcher indica il valore di ΔT espresso in ms. Tutti gli esperimenti sono ripetuti per 10 prove, i risultati presentati sono da intendersi quindi mediati su di esse (i valori del "Massimo" di tabella 4 sono invece valori assoluti).

	Direct Call	Thread	PLC	Dispatcher 10	Dispatcher 0.1
Media	43,863 ms	43,478 ms	46,136 ms	46,846 ms	45,932 ms
Varianza	0,207 ms	0,119 ms	0,295 ms	0,307 ms	0,308 ms
Massimo	44,132 ms	44,966 ms	47,256 ms	54,935 ms	56,176 ms

Tabella 4 - Tempo di esecuzione del singolo blocco funzionale

Il *secondo esperimento* consta nella valutazione di particolari intervalli temporali in una applicazione in cui *n* istanze del blocco funzionale vengono collegate in "serie", in cui l'evento *COLD* del blocco *E_RESTART* è connesso con l'evento *REQ* del primo blocco funzionale *LOOP* e l'evento *CNF* del blocco *LOOP i*-esimo è connesso con l'evento *REQ* del blocco *LOOP i+1* .
Il primo intervallo temporale di interesse è il tempo medio di esecuzione del blocco funzionale definito come:

(a) $$\overline{t_{FB}} = \frac{1}{n}\sum_{i=1}^{n}\left(t8(i) - t2^{+}(i)\right)$$

I valori medi di t_{FB} nel caso di applicazioni con n (n = 1…100) blocchi funzionali in "serie" sono mostrati nella figura 4. Essa mostra che, in un'applicazione con un numero di blocchi maggiore di 20 per gli approcci Direct Call, Thread e PLC, il valore si mantiene circa costante e in media è di circa 32 ms mentre è di 38 ms per l'implementazione Dispatcher.

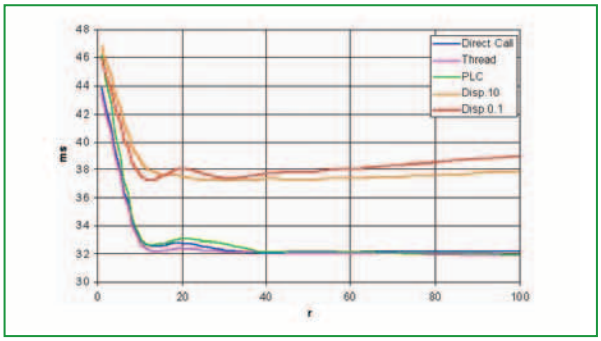


Figura 4 - Tempo medio di esecuzione di un blocco funzionale (applicazione "serie")

Il secondo intervallo temporale considerato è il tempo medio di esecuzione di una connessione eventi definito come:

$$\overline{t_{CON}} = \frac{1}{n-1}\sum_{i=2}^{n}\left(t2^{-}(i) - t8(i-1)\right)$$

I risultati dell'esperimento per i valori medi di t_{CON} sono mostrati nella figura 5. I valori per gli approcci PLC e Direct Call sono simili e costanti, in media 15 µs per il primo e 20 µs per il secondo. Leggermente superiore è il valore medio per l'approccio Thread (circa 40 µs). Risulta essere invece molto alto tale intervallo nel caso del Dispatcher (circa 4,4 ms).

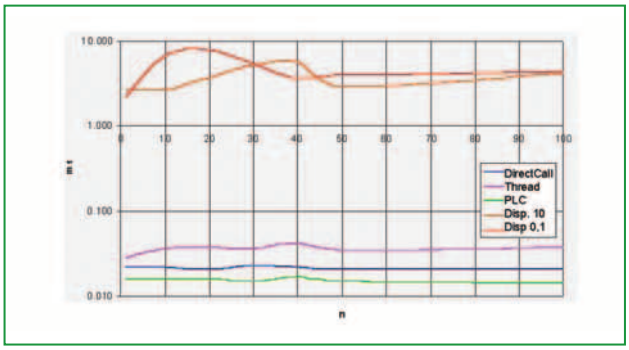


Figura 5 - Tempo medio di esecuzione di una connessione eventi (applicazione "serie")

Il *terzo esperimento* prevede di collegare i blocchi funzionali in "parallelo", in cui l'evento *COLD* del blocco *E_RESTART* è connesso con l'evento *REQ* di *m* blocchi funzionali *LOOP*.
Anche in questo caso il primo intervallo di tempo misurato è il tempo medio di esecuzione di un blocco funzionale definito in (a) (per *m* blocchi funzionali).
I risultati di questo esperimento sono rappresentati nella figura 6. Essa mostra che il tempo di esecuzione di un blocco funzionale rimane pressoché invariato (rispetto al caso "serie") e costante per le implementazioni Direct Call e PLC mentre cresce al crescere del numero di blocchi funzionali nella soluzione Thread e ancor di più nella soluzione Dispatcher. Questo è evidente poiché, contrariamente alle prime due, queste ultime soluzioni prevedono una esecuzione in parallelo (regolata nel caso Dispatcher) dei blocchi attivi.
Le prestazioni dei metodi Direct Call, Thread e Dispatcher si invertono se si considera, invece del tempo medio di esecu-

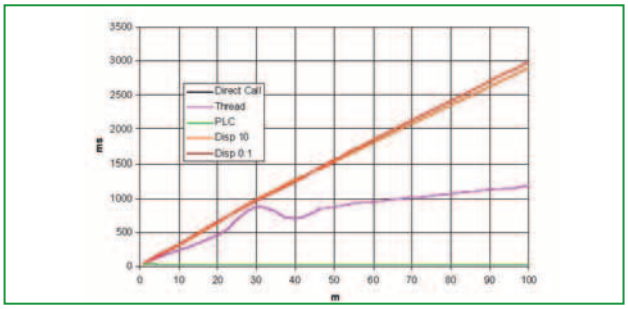


Figura 6 - Tempo medio di esecuzione di un blocco funzionale (applicazione "parallelo")

zione del blocco funzionale, il tempo di esecuzione di una connessione eventi definita come:

$$t_{CON}(i) = t2^{-}(i) - t8(START) \quad \text{con } i = 1..m \quad (m = 100)$$

Il grafico della figura 7 mostra infatti l'andamento di questo valore in funzione della "posizione" della connessione, intendendo in posizione 1 la connessione tra *START* e il primo blocco funzionale *LOOP*, in posizione 2 la connessione tra *START* e il secondo blocco funzionale *LOOP* ecc. La figura conferma il comportamento indesiderato dell'implementazione Direct Call già sottolineato nel terzo paragrafo. Nel grafico i valori per gli approcci PLC e Thread sono così piccoli da essere difficilmente visualizzabili.

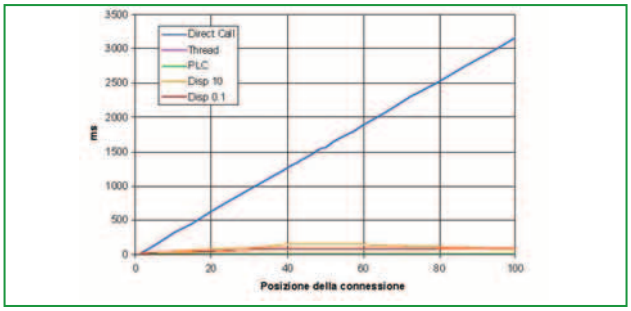


Figura 7 - T_{CON} in funzione della "posizione" della connessione (applicazione "parallelo")

Dai grafici della figura 6 e figura 7 sembrerebbe che l'implementazione PLC sia la migliore, in realtà c'è un altro intervallo di tempo da considerare in cui questo approccio mostra i suoi limiti. L'intervallo di tempo in questione è il tempo T_{wake} prima definito. Nell'approccio PLC, come del resto in quello Dispatcher, questo tempo dipende dal tempo di ciclo della scansione e risulta quindi essere tanto maggiore quanto maggiore è il numero di blocchi funzionali.
L'ultimo esperimento vuole raffrontare il tempo di esecuzione di una applicazione formata da *n x m* blocchi funzionali connessi come mostrato nella figura 8. Tale tempo è definito come:

$$t_{APP}(n,m) = \max_{i=1..m}\{t8(in)\} - t8(START)$$

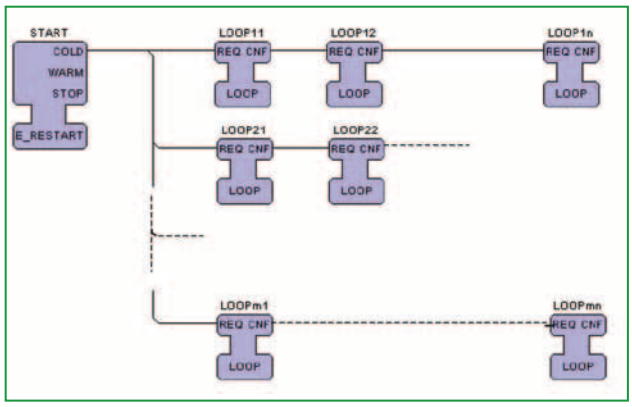


Figura 8 - Un'applicazione "serie-parallelo"

I risultati di questo esperimento sono riportati nella figura 9. L'approccio Thread dà il minor tempo di esecuzione. Leggermente più alto è il tempo per l'approccio Direct Call. L'approccio PLC è invece più lento di circa il 18% mentre molto più lento è l'approccio Dispatcher.

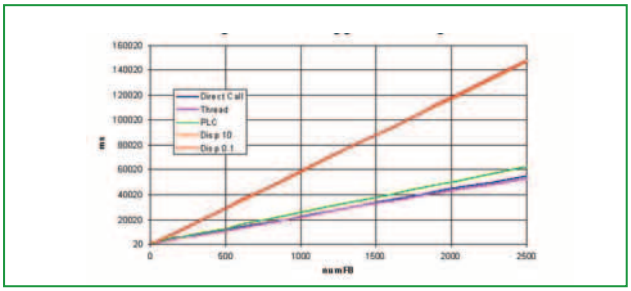


Figura 9 - Tempo di esecuzione di un'applicazione di n × m blocchi funzionali

È ovvio che questo è solo un caso particolare e che questi risultati offrono solo un'idea qualitativa in quanto molto dipende dal tipo di blocchi funzionali, dalla tipologia dell'applicazione, e dalla piattaforma HW/SW utilizzata.
Dagli esperimenti fatti si può concludere che il metodo:

- Direct Call: è la soluzione migliore in presenza di "serie" di blocchi funzionali; presenta un comportamento indesiderato quando più eventi devono essere generati allo stesso tempo; presenta un comportamento indesiderato in presenza di retroazioni (l'evento *CNF* è connesso con l'evento *REQ* dello stesso blocco).
- Thread: segue il comportamento desiderato e l'idea di blocchi paralleli; presenta un tempo di esecuzione del blocco funzionale fortemente dipendente dal numero di blocchi attivi.
- PLC: permette di fare una previsione sulle performance temporali di un'applicazione; presenta un tempo di esecuzione del blocco funzionale fortemente dipendente dall'ordine in cui vengono schedulati i blocchi.
- Dispatcher: presenta un'esecuzione parallela (virtuale) dei blocchi; ha prestazioni temporali dipendenti dall'ordine della sequenza in cui vengono schedulati i blocchi e dal va-

lore di ΔT (permette di fare una previsione sulle prestazioni temporali di un'applicazione).

Esempio applicativo

L'esempio seguente mostra l'utilizzo della normativa IEC 61499 per affrontare un problema di controllo del moto "classico", come è quello di controllo della posizione angolare di un giunto robotico [3].
Il modello del giunto utilizzato è un modello che non tiene conto dei fenomeni di elasticità ed è semplificato in alcune sue componenti. Il controllo utilizzato è invece un classico controllo in cascata con anello interno di velocità regolato da un PI e un anello esterno di posizione regolato da un semplice guadagno.
Il modello del sistema controllato è mostrato nella figura 10, dove il modello del giunto è contenuto all'interno della linea rossa tratteggiata e dove Θ_r è il valore di riferimento, Θ rappresenta la posizione angolare, la sua derivata prima la velocità angolare, la sua derivata seconda l'accelerazione angolare, J è il momento d'inerzia e D_m è il coefficiente di attrito viscoso.

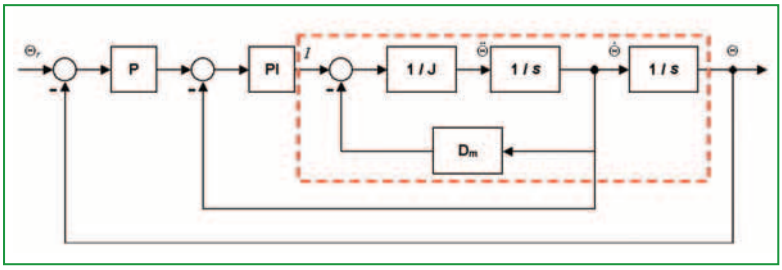


Figura 10 - Schema di controllo in cascata posizione-velocità del giunto

La rappresentazione a blocchi funzionali secondo la IEC 61499 del modello della figura 10 è mostrata nella figura 11 in cui è possibile riconoscere i blocchi funzionali che modellizzano il giunto robotico e i blocchi che realizzano il controllo.

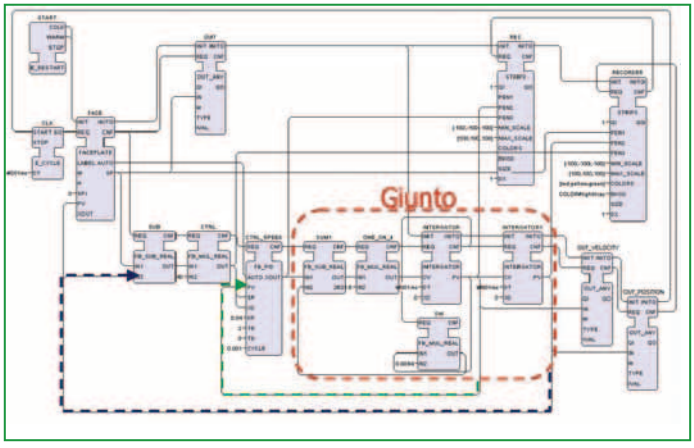


Figura 11 - Modello a blocchi funzionali del controllo del giunto

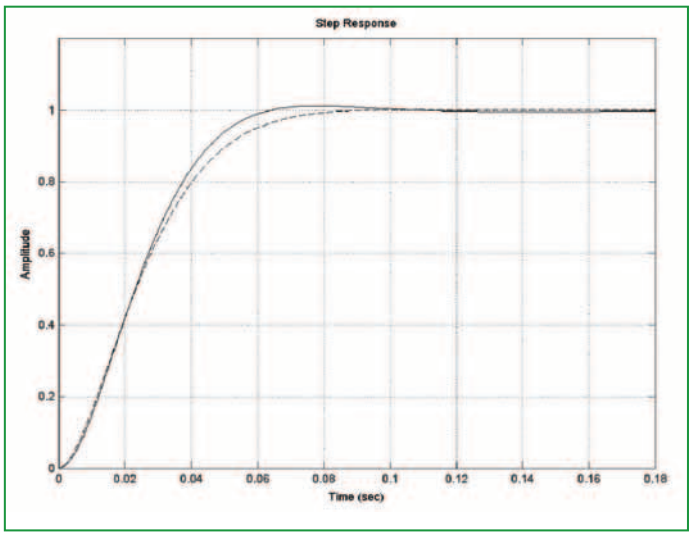


Figura 12 - Risposta a scalino del modello in Matlab-Simulink (linea continua) e del modello a blocchi funzionali (linea tratteggiata)

Quest'applicazione implementata seguendo l'approccio Thread permette di avere una risposta a scalino molto vicina a quella data dallo stesso modello della figura 10 utilizzando il pacchetto software Matlab-Simulink, come mostrato nella figura 12.
Lo stesso sistema può essere distribuito su due risorse differenti in modo da separare il controllo dal modello del giunto. Questa separazione viene fatta aggiungendo dei blocchi detti di comunicazione (Publish e Subscribe) che permettono alle due risorse di scambiare messaggi (per maggiori dettagli consultare [3, 5] o direttamente [7]).
È ora interessante effettuare delle prove sperimentali per valutare se il sistema così realizzato può rappresentare una simulazione corretta di un sistema reale. Nel caso dell'applicazione centralizzata infatti il tempo di simulazione è imposto a 1 ms, uguale al passo di discretizzazione. Il vincolo temporale da soddisfare risulta quindi che l'intera applicazione (controllo e modello) sia eseguita in meno di 1 ms. La stessa prova è stata eseguita con sei applicazioni di controllo di un giunto in parallelo, a rappresentare il controllo del moto di un manipolatore completo. Le prove in questione sono state effettuate sia per l'implementazione Thread che per quella realizzata da Fbdk. Si è inoltre testato il funzionamento di questa applicazione distribuita quando questa risiede su risorse diverse dello stesso dispositivo e risorse diverse su dispositivi diversi collegati attraverso una rete Ethernet/IP a 100 Mb/s di 3 m di lunghezza.
La figura 13 mostra il tempo di esecuzione del modello del giunto, del controllo e tempo totale nel caso in cui l'applicazione risieda su una singola risorsa, su due risorse di uno stesso dispositivo, su dispositivi diversi connessi tramite rete Ethernet/IP; le prove sono state fatte per un'applicazione con un singolo giunto (in alto)

e con 6 giunti (in basso).
Dagli esperimenti fatti, si può notare che il tempo di calcolo dell'applicazione su una risorsa per le due versioni Fbdk ("Call" nella figura 13) e Thread è inferiore a 0,1 ms. Nel caso di applicazione distribuita sullo stesso dispositivo ma su risorse diverse al tempo di calcolo dell'algoritmo si aggiunge il tempo di comunicazione locale.
Il tempo così risultante è per entrambe le versioni circa 0,1 ms. Nel caso di applicazione distribuita su dispositivi diversi al tempo di calcolo dell'algoritmo si aggiunge il tempo di comunicazione in rete.
Il vincolo da soddisfare in questo caso è che sia controllo che modello, compresi i tempi di comunicazione, non superino il valore di 1 ms.
Il tempo risultato dalle prove sperimentali, nel caso qui descritto è per entrambe le versioni di circa 0,75 ms per il modello del giunto e 0,72 ms per il controllo. Le prove con 6 applicazioni di controllo e simulazione in parallelo mostrano, nei casi di applicazione su una risorsa e di applicazione su più risorse sullo stesso dispositivo, un aumento proporzionale del tempo di calcolo complessivo.
La prova distribuita mostra invece che i tempi di comunicazione di rete non si sommano, dato che si accede alla rete da porte diverse, e anche in questo caso sia modello che controllo rispettano il vincolo di 1 ms.

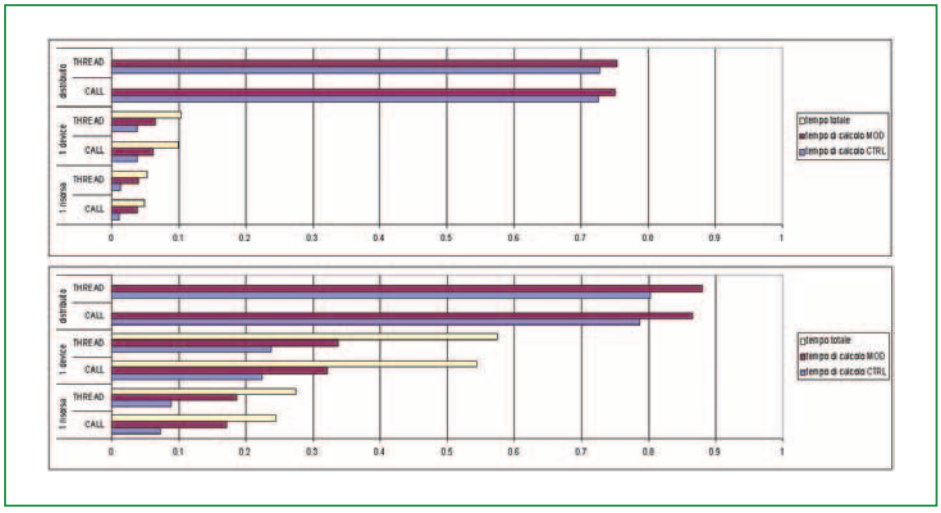


Figura 13 - Confronto tra i valori temporali registrati per i diversi esperimenti per uno (in alto) e per 6 giunti (in basso)

Considerazioni conclusive

L'implementazione di un modello astratto usato per la programmazione di applicazioni di automazione distribuite come quelle della normativa IEC 61499 è molto importante per determinare e predirne il comportamento temporale. Tali considerazioni sono molto importanti per capire il reale campo di applicabilità di questa normativa.
In questo documento sono stati mostrati e analizzati alcuni possibili approcci implementativi ed è stato descritto brevemente l'approccio utilizzato da un software esistente chiamato Fbdk.
Lavori futuri prenderanno in considerazione la possibilità di estendere la normativa in modo da poter definire dei vincoli real-time nelle applicazioni di controllo e quindi di poterli testare o verificare già in fase di progetto.
Seguendo questi concetti verranno inoltre studiati tutti quei meccanismi e quelle estensioni che potranno permettere in futuro l'utilizzo di Java anche per sistemi real-time.

Bibliografia

[1] R. David, "Grafcet: A powerful tool for specification of logic controllers", *Ieee Transactions on Control Systems Technology*, n. 3, pp. 253-268. 1995.
[2] L. Ferrarini, E. Carpanzano, "A Structured Methodology for the Design and Implementation of Control and Supervision Systems for Robotic Applications", *Ieee Transactions on Control System Technology*, vol. 10, n. 2, pp. 272-279, Marzo 2002.
[3] L. Ferrarini, C. Veber, *IEC 61499 - Uno standard per sistemi distribuiti di automazione industriale*, Pitagora Editrice Bologna, 2004.
[4] Holobloc.com, "Function Block-Based", *Holonic Systems Technology*, sito web: www.holobloc.com.
[5] R. Lewis, "Modelling control systems using IEC 61499", *Ieee control engineering series*, n. 59, The Institution of Electrical Engineers, Londra, UK, 2001.
[6] C. Schwab, M. Tangermann, A. Lüder, A. Kalogeras, L. Ferrarini, "Mapping of IEC 61499 Function Blocks to Automation Protocols within the Torero Approach", Indin04, *Ieee International Conference on Industrial Informatics*, 24-26 giugno 2004, Berlino, Germania, 2004.
[7] Standard IEC 61499, *Function Blocks for Industrial-Process Measurements and Control System*, IEC TC65/WG6, Parte I, PAS, 2001.
[8] V. Vyatkin, "Intelligent mechatronic components: control system engineering using an open distributed architecture", *Emerging Technologies and Factory Automation*, 2003. *Proceedings. Etfa '03, Ieee Conference*, Vol. 2 , pp. 277 – 284, 16-19 settembre 2003.
[9] W.E. Rumpl, F. Auinger, C. Dutzler, A. Zoitl, "Platforms for Scalable Flexible Automation Considering the Concepts of IEC 61499", *Ifip/Ieee International Conference on Information Technology for Balanced Automation Systems in Manufacturing and Services*, Cancun, Messico, pp. 237-246, 25-27 settembre 2002.