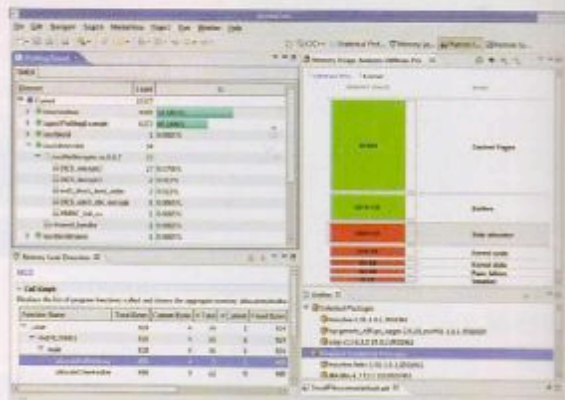


# La produttività nello sviluppo degli ambienti Linux

Solo con gli strumenti adeguati è possibile migliorare la produttività nello sviluppo delle applicazioni embedded basate sul sistema operativo Linux

**Jim Ready**  
Fondatore e CTO  
MontaVista Software

**Bill Weinberg**  
General manager  
Linux Phone Standards Forum



**Fig. 1 - Immagine del DevRocket 5.0 che mostra la possibilità di creare istogrammi utili per analizzare e verificare l'utilizzo delle memorie**

**P**er ragioni culturali, tecniche e storiche, gli sviluppatori Linux hanno ormai imparato a lavorare con tool piuttosto spartani e basati su linee di comando alquanto scomode, note come Command Line Interface, CLI. Oggi è ben difficile riscontrare una significativa diffusione di sofisticati Integrated Development Environment (IDE) fra gli sviluppatori di applicazioni Linux, convinti più che mai che "un bravo sistemista Linux detesta i tool IDE", come recita un ricorrente slogan. Di fatto, l'approccio CLI nello sviluppo delle applicazioni embedded è divenuto un uso e costume ormai consolidato. Tuttavia, l'incessante avanzare della Legge di Moore e il continuo complicarsi delle piattaforme di elaborazione embedded costringono i progettisti a realizzare software embedded sempre più voluminosi e multifunzionali. Di conseguenza comincia a rendersi maggiormente evidente la difficoltà di dover affrontare fasi di sviluppo che con i tradizionali metodi CLI diventano davvero onerose e ciò si traduce sempre più spesso in accumuli di ritardo nella messa a punto delle nuove applicazioni e, dunque, in perdite di time-to-market. D'altra parte, talvolta anche la scelta di usare un IDE può

essere deleteria, giacché può finire per comportare time-to-market ancora più lunghi e, quindi, peggiorare ulteriormente la situazione.

A questo punto è lecito chiedersi se c'è qualche realistica alternativa ai tool CLI per lo sviluppo delle applicazioni embedded basate su Linux. Di fatto, ci sono molte aziende che sarebbero certamente disposte a investire tempo e risorse in progetti Roll Your Own, RYO, ossia finalizzati a realizzare piattaforme di sviluppo Linux "open source" personalizzabili e dotate di funzionalità in tempo reale. Al contrario, poche aziende accetterebbero di partecipare a progetti RYO già avviati da qualcun altro. Questa riluttanza deriva dall'evidente osservazione che i tool FOSS, Free and Open-Source Software, in circolazione non sono giudicati con favore rispetto ai tool commerciali che accompagnano i sistemi operativi RTOS sul mercato. In altre parole, i progetti RYO mancano proprio dei requisiti fonda-

```

vma samples & symbol name
001c974 5016 0.5096 _Rb_brcunassigned short, pair(unsigned short const, int),
unsigned short const>::find(unsigned short const&)
0010c40c 3020 5.6375 Paragraph::getFontSettings(BufferParams const&, int) const
00131940 3220 5.4627 LyXText::getFont(Buffer const&, Paragraph*, int) const
000e4540 3011 5.1082 LyXFont::realise(LyXFont const&)
000e3d70 2020 4.9497 LyXFont::LyXFont()
001255a0 1020 0.0927 LyXText::singleWidth(BufferView*, Paragraph*, int, char) const
000a2c00 1004 0.0605 operator==(LyXFont::FontBites const&,
LyXFont::FontBites const&)
001120c0 1729 2.9332 Paragraph::Pimpl::getChar(int) const
001ab094 1027 2.0816 qfont_loader::getFontInfo(LyXFont const&)
#

```

Fig. 2 - Esempio di codice in uscita dal tool OProfile

mentali riguardanti il livello d'integrazione e la facilità d'uso. Questi sono, a ben vedere, i motivi del successo dei tool MontaVista, reputati popolarmente come semplici da usare quanto potenti nelle funzionalità e oggi integrati nell'ultima versione del DevRocket 5.0 IDE, capace di offrire al progettista un'esperienza di sviluppo sorprendente per molti aspetti.

Inoltre, i tool IDE proprietari che accompagnano talvolta gli RTOS difettano proprio nelle caratteristiche che più servono allo sviluppo delle applicazioni software basate su Linux. Come riportato nella tabella 1, fra le insufficienze più evidenti ci sono la mancanza di supporto per più linguaggi e la troppo stretta dipendenza dalle funzionalità Windows, due limiti che generalmente finiscono per costringere gli sviluppatori a legarsi a un singolo fornitore di tool IDE.

### I requisiti indispensabili per un IDE Linux

Per essere davvero utile, un moderno IDE deve essere molto di più che un semplice editor di linee di programma con sup-

porto alla loro compilazione e debug. MontaVista ha, infatti, realizzato il tool Eclipse pensando proprio alle esigenze dei progettisti e integrando insieme algoritmi open-source, tool proprietari e strumenti specifici per l'analisi, il debug e l'ottimizzazione dei codici.

Per esempio, MPatrol è un tool attualmente molto popolare per la sua flessibilità però non si tratta di un vero e proprio strumento, ma di una libreria di algoritmi run-time molto utili per effettuare diagnosi sugli errori di programmazione causati dal non corretto uso delle istruzioni di allocazione dinamica della memoria. In pratica, il pregio dell'MPatrol è che può essere indirizzato da un altro programma oppure richiamato da un tool o, ancora, utilizzato dal debugger come malloc() nell'allocazione dinamica della memoria, così da servire alle API per tracciare le istruzioni di chiamata malloc() e free(). Sebbene sia estremamente utile, tuttavia, MPatrol non è un vero e proprio debugger integrato, perché costringe sempre a un po' di lavoro per aggiustarne le prestazioni e adattare le funzionalità alle applicazioni.

Non solo, ma dopo aver faticato un po' nel lavoro di adattamento, bisogna per forza almeno verificare l'uso della memoria e tracciare i trasferimenti dati.

Aggiungendo MPatrol nell'Application Developer Kit, APK, dell'IDE DevRocket come componente solidamente integrata, MontaVista ha realizzato una potente piattaforma di sviluppo, pur dotata di risorse open-source (Fig. 1).

Pertanto, le funzioni di debug sulle memorie incorporate nell'ADK MontaVista offrono agli sviluppatori i seguenti vantaggi: uso trasparente di tutti gli strumenti sui codici applicativi; avvertimenti ed errori vengono segnalati direttamente sull'IDE e non riversati in file poco visibili; sono generate automaticamente sia le chiamate grafiche sia le istruzioni di chiamata da e verso la memoria; c'è la possibilità di tracciare l'uso della memoria durante l'esecuzione degli algoritmi; si possono rintracciare i collegamenti fra le istruzioni del codice eseguibile e le chiamate grafiche.

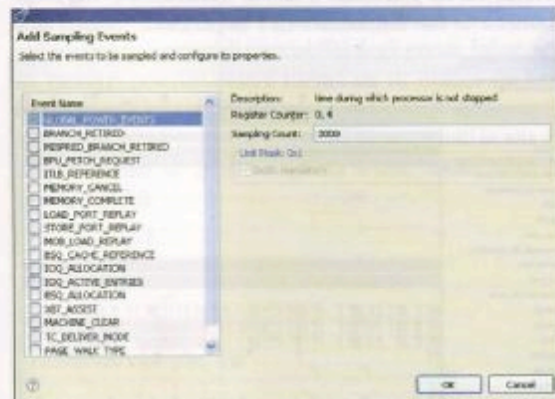


Fig. 3 - Esempio di configurazione dei controlli per il campionamento degli eventi sull'IDE DevRocket 5.0