



Ambienti aperti di sviluppo software, per Linux e non solo

Gli strumenti per creare applicazioni e componenti del sistema operativo. Come scegliere, tra le tante offerte, quella adeguata agli scopi.

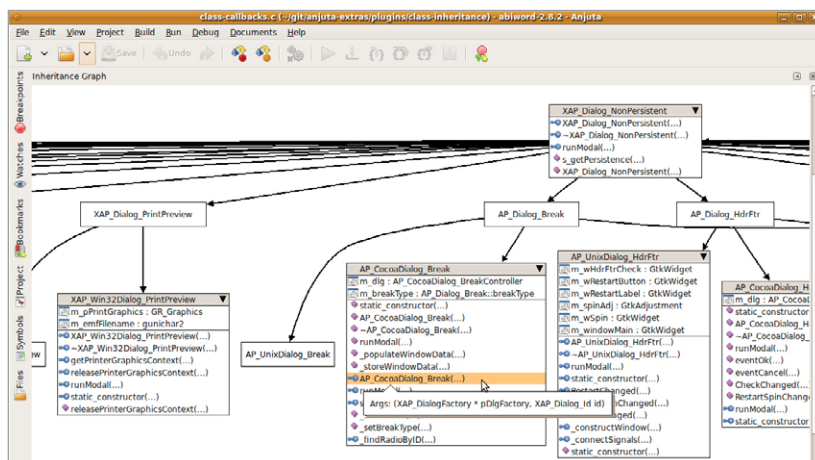
Come e dove si crea un programma per Linux, dal kernel in su, partendo da zero? Con quali strumenti lo si può modificare e collaudare, per continuare a migliorarlo? Il sistema tradizionale per svolgere queste attività è ancora perfettamente funzionante e molto popolare, almeno in certi ambienti o per certe categorie di software: scrivere codice con editor venerandi come Emacs o Vi, poi compilarlo e validarlo interamente da riga di comando, decifrando direttamente su terminale i messaggi diagnostici di strumenti come Gcc o Gdb. Per chi, comprensibilmente, non volesse saperne di questo modo di lavorare esistono da un pezzo diverse alternative molto valide: Ide (*Integrated Development Environment*), cioè ambienti grafici di sviluppo software, molto completi e interamente Open Source. Dietro le quinte questi sistemi usano gli stessi strumenti (ovviamente aggiornati) di quindici o vent'anni fa, ma lo fanno in un modo che facilita moltissimo il lavoro, presentando un'unica interfaccia per tutte le fasi dello sviluppo e provvedendo da soli ai compiti più banali e noiosi. In questo numero presentiamo alcuni degli ambienti più interessanti di sviluppo software per *desktop* Linux. Il mese prossimo, invece, ci occuperemo di quelli per lo sviluppo di applicazioni Web. Per facilitare la lettura è opportuno iniziare da una brevissima descrizione delle principali operazioni di creazione software in ambiente Gnu/Linux.

Compilazione, linking, debugging...

Il famigerato "codice sorgente" la cui piena disponibilità è alla base del mondo Open Source, è una descrizione del funzionamento di un programma, scritta in un qualche linguaggio inventato appositamente per questo scopo. Questi linguaggi sono grossolanamente divisibili in due categorie. Nella prima (linguaggi interpretati), il codice sorgente viene letto e direttamente eseguito, una linea alla volta, da un apposito *interprete*. Fra quelli attualmente più popolari in ambito Open Source troviamo Python, Perl, Ruby e gli inossidabili script shell. I linguaggi *compilati*, invece, sono quelli che devono essere tradotti in blocco in file chiamati spesso binari o eseguibili, perché sono sequenze di istruzioni in linguaggio macchina immediatamente comprensibili ad un microprocessore. Questo è il compito di applicativi specializzati, chiamati appunto compilatori, che provvedono anche al *linking*, cioè al collegamento

fra il nucleo di un programma e tutte le librerie di sistema di cui potrebbe aver bisogno. I programmi in linguaggi compilati, come C o C++, sono molto più complessi da scrivere di quelli interpretati. Chi li scrive deve occuparsi personalmente di gestione della memoria, dichiarazione variabili e mille altre cose che nell'altro caso sono a carico dell'interprete. Il vantaggio è una molto maggiore flessibilità e, quasi sempre, rapidità di esecuzione.

La scelta dei comandi di compilazione migliori per un dato programma, e delle opzioni più adatte per ognuno di loro, non è la parte più complessa dello sviluppo software. Però è quella quella in cui, almeno per un principiante, è più facile fare errori e perdere, per così dire, il controllo della situazione. In generale, queste istruzioni dipendono almeno da quale combinazione di processore e sistema operativo (o distribuzione, nel caso di Linux) si vuole servire: le stesse librerie potrebbero trovarsi in directory diverse da sistema a sistema. Anche la *versione* di programma da creare, per

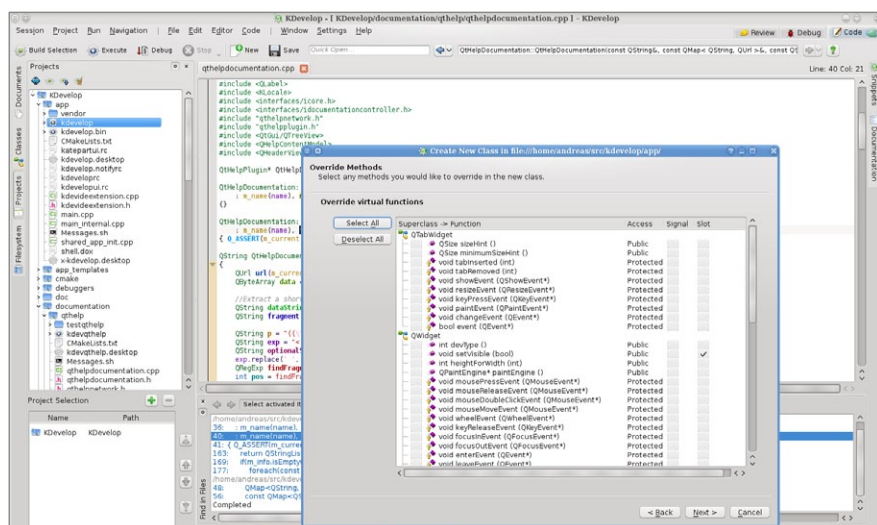


Scrivere codice velocemente serve a poco, se non si possono controllare facilmente proprietà e relazioni di tutte le funzioni e classi di oggetti. Anjuta risolve il problema con questo e altri grafici.

Passare da un ambiente all'altro con EditorConfig

Provare più ambienti di sviluppo finché non si scopre quello più adatto ai propri gusti ed esigenze può richiedere parecchio tempo, dal momento che ogni sistema ha i suoi file e il suo stile di configurazione. L'utilità EditorConfig (<http://editorconfig.org>) è nata proprio per risparmiare più tempo possibile quando si vuole passare spesso da un ambiente all'altro (o si è costretti a farlo, magari perché si lavora come consulenti per diverse aziende allo stesso tempo). Con EditorConfig basta impostare le proprie preferenze e stile di codifica in un solo file di configurazione. Quel programma provvederà poi da solo a generare file di configurazione corrispondenti a diverse Ide. I plugin di EditorConfig consentono di applicare le stesse impostazioni anche a diversi editor per programmatori. Inoltre, essendo semplici file di testo possono essere gestiti senza problemi dai normali sistemi di controllo delle revisioni usati per il codice sorgente.

sviluppo gestito con Anjuta sono quelli per la gestione di file e progetti. Il file manager funziona come quelli normalmente utilizzati per orientarsi fra file e cartelle del proprio computer. I menu contengono però anche tutte le azioni specifiche per sviluppo e debugging di software: compilazione, generazione di nuove versioni del prodotto e così via. Il project manager fornisce un'interfaccia grafica unificata ai sistemi a basso livello di configurazione, generazione e uso di Makefile cui abbiamo già accennato. Ciò che l'utente vede è un'unica gerarchia di oggetti, divisi in gruppi e target. I primi corrispondono a cartelle di file sorgente, i secondi alle operazioni di configurazione (Makefile e file collegati) relative a quelle stesse cartelle. Secondo gli sviluppatori Anjuta può lavorare come front-end di qualsiasi sistema di configurazione e Makefile "del pianeta". Certo, provare al di là di ogni dubbio un'affermazione così impegnativa è difficilissimo. D'altra parte, al di là dei proclami più o meno pubblicitari, quello che è davvero utile è interessante è il motivo, cioè la



Come Anjuta, anche Kdevelop offre archivi facilmente consultabili di tutte le classi di oggetti nel progetto.

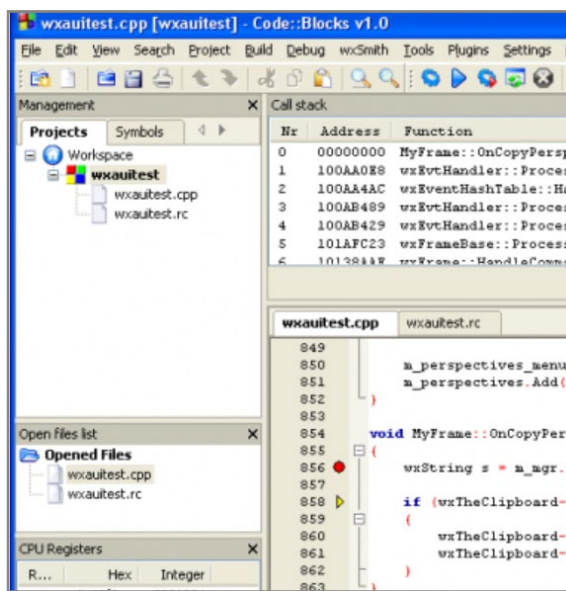
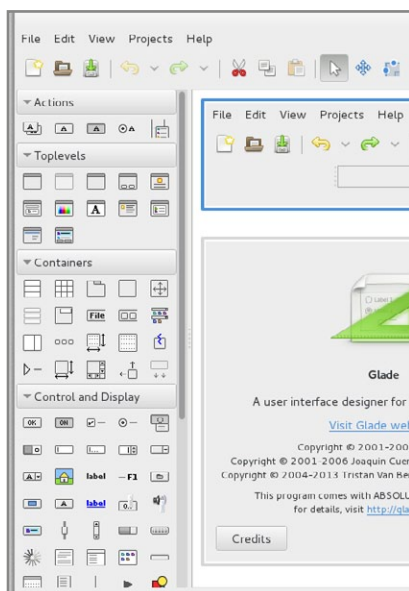
esempio demo, completa, con o senza chiave unica associata alla licenza eccetera, fa differenza. Per fare tutto questo in modo corretto è necessario innanzitutto configurare le istruzioni di compilazione, cioè adattare all'ambiente in cui il programma verrà eseguito (che potrebbe essere completamente diverso da quello su cui è sviluppato e compilato). Il risultato della compilazione è la creazione di una qualche sorta di *Makefile*. Questo è il nome tradizionalmente usato per indicare al compilatore non solo cosa deve fare (in inglese, appunto, *make*), ma soprattutto in quale ordine e in quali casi. I programmi più complessi sono infatti composti da parecchie decine di migliaia di linee di codice, sparse per comodità di lavoro in centinaia o migliaia di file. Se non venissero caricati nell'ordine giusto, il compilatore riceverebbe istruzioni di utilizzare nel programma finale funzioni o variabili di cui ancora non sa nulla. I Makefile sono necessari anche per ridurre i tempi di lavoro. Se si è cambiata una sola riga in un solo file (caso comunissimo) perché ricompilare tutti i sorgenti, cosa che per certe applicazioni potrebbe richiedere ore? Molto meglio elaborare solo il file modificato e tutti quelli che effettivamente dipendono, direttamente o meno, dal suo contenuto. Prevedibilmente, la storia non finisce qui. Il *debugging*, cioè l'individuazione dei bachi (bug) nel codice, normalmente porta via molto più tempo della sua scrittura. Non è facile scoprire da quale istruzione è stato causato un errore, o quale era il valore di tutte le variabili

in quel momento. Infine, è anche necessario tenere sotto controllo strettissimo, soprattutto quando si lavora in gruppo, quale versione di ogni singolo file sorgente, chiunque lo abbia scritto, è stata usata nella compilazione di ogni versione eseguibile. A questo provvedono i sistemi di controllo delle revisioni, su cui torneremo il prossimo mese.

Anjuta

Anjuta DevStudio (<https://wiki.gnome.org/Apps/Anjuta>) è una Ide molto ricca, nata nella comunità del desktop Gnome, che cerca di fornire tutto quel che potrebbe essere utile a un programmatore con la massima semplicità possibile. I tanti pannelli (anche decine) che potrebbero servire per uno sviluppo complesso sono collocabili a piacere nella finestra principale e sempre minimizzabili con un clic. All'interno di ogni progetto posizione e dimensioni di ogni pannello, che potrebbero anche cambiare nel corso del lavoro, sono memorizzate insieme a tutti gli altri parametri di configurazione. Lo stesso vale per il contenuto di alcuni dei menu del programma. In generale, Anjuta funziona integrando plugin dei tipi più disparati, permettendo all'utente di attivarli o disattivarli anche nel corso della stessa sessione, senza dover riavviare l'applicazione. Quando sono disponibili più plugin per una stessa funzione sta all'utente impostare manualmente quello che dovrà essere utilizzato. In pratica, gli unici plugin con cui si avrà sicuramente a che fare in qualsiasi

Un momento di lavoro con Glade: qui vediamo i controlli con cui costruire un'interfaccia grafica con pochi clic del mouse.



Anjuta e Kdevelop sono le Ide Open Source più popolari, ma non certo le uniche né le migliori in tutti i casi. CodeBlocks, per esempio, gira senza problemi anche su OS X e Windows.

caratteristica di Anjuta, per cui vengono fatti: la mancanza di un formato di configurazione specifico. Infatti, pur fornendo un'interfaccia unificata a qualsiasi sistema di compilazione a basso livello, Anjuta non modifica affatto con opzioni non standard i relativi file, né scrive altrove le varie impostazioni del progetto corrispondente. Questo permette sia di caricare senza problemi lavori originariamente sviluppati in altri ambienti sia, soprattutto, di lavorare anche con partner che non usano Anjuta. L'unica regola da rispettare al riguardo, che sarebbe valida anche se non si usasse Anjuta, è seguire rigorosamente le buone pratiche standard consigliate per ogni flusso Make.

Oltre ai plugin, Anjuta integra anche non pochi programmi completi, utilizzabili autonomamente anche da chi non ha bisogno di una Ide vera e propria. L'esempio migliore è l'editing del codice, possibile sia con Scintilla (www.scintilla.org) sia con GtkSourceView (<http://wiki.gnome.org/Projects/GtkSourceView>). Nella versione visibile in Anjuta, entrambi i programmi funzionano sostanzialmente nello stesso modo: colorizzazione della sintassi, collassamento (*code folding*) del codice di ogni funzione per visualizzare in un'unica finestra una parte maggiore del programma e soprattutto Calltips (alla lettera "suggerimenti di chiamata"): questo è il nome che Anjuta dà alle mini-finestre pop-up che appaiono quando si digita il nome di una funzione software. I calltips velocizzano la scrittura di codice e riducono gli errori perché mostrano quali parametri si possono assegnare alla funzione corrente. Durante i collaudi, gli stessi editor riposizionano automaticamente il cursore sulla riga

di codice corrispondente all'errore segnalato dal debugger.

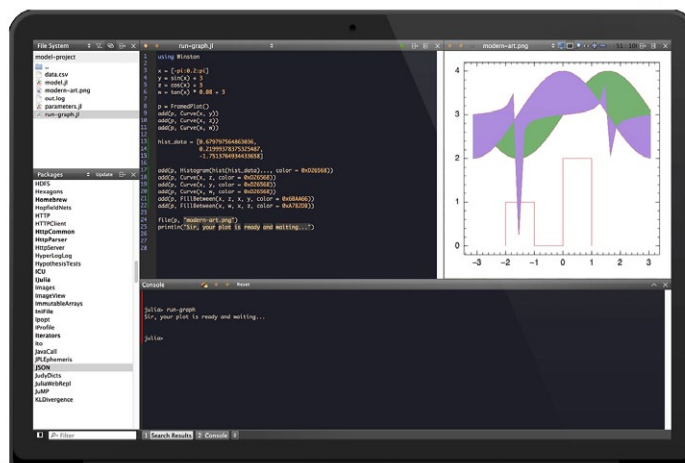
Il sistema di sviluppo grafico rapido chiamato Glade (<https://glade.gnome.org>) è normalmente utilizzato nel progetto Gnome per creare interfacce a finestre con menu e pulsanti basati sulle librerie Gtk. Come gli editor, anche Glade è utilizzabile sia da solo sia come componente perfettamente integrato di Anjuta. Parametri e comandi Glade per creare i componenti grafici di un programma a finestre vengono salvati in file di testo Xml. Anjuta permette di modificare questi ultimi anche nel suo editor, se necessario, ma normalmente è sufficiente fargli eseguire Glade all'interno della finestra principale.

Anche il browser per documentazione software di Gnome, chiamato Devhelp, è inserito in Anjuta per facilitare la consultazione di tutte le descrizioni di funzioni e simboli software.

Altri plugin accelerano la creazione di classi C++, modelli per file sorgente e

visualizzazione di tutte le classi utilizzate in ogni progetto, arrangiandole gerarchicamente per mostrare le relazioni fra l'una e l'altra.

Terminata la scrittura del codice, Anjuta offre un'interfaccia completa al debugger standard del progetto Gnu, gdb (www.sourceware.org/gdb): è lui a gestire tutte le operazioni standard per questo tipo di attività, dai timer all'impostazione di breakpoint, che sono punti dove bloccare lo svolgimento del programma per analizzarne lo stato. Il debugger di Anjuta può controllare l'attuazione del codice anche in altri modi: si può eseguire un'istruzione alla volta o passare direttamente a un punto generico indicato dal programmatore. Nessun problema per i dump di memoria, che sono copie complete, dalla Ram al disco rigido, dello stato di tutti i registri e variabili interne di un programma. L'analisi è completata dall'integrazione di Anjuta con Valgrind, il profiler descritto a fine articolo.



Julia Studio, una Ide sviluppata appositamente per compilare programmi di calcolo numerico scritti nel linguaggio Julia, permette anche di esaminare i grafici da essi generati in tempo reale o quasi.

Kdevelop, l'ambiente ottimale per Kde

L'altro Ide Open Source che va per la maggiore in ambiente Linux è KDevelop (<http://kdevelop.org>), simile ad Anjuta per impostazione e funzionalità generali, ma con due differenze importanti: prima di tutto, Anjuta è prevalentemente pensato per lo sviluppo di applicazioni C per Gnome. KDevelop, invece, è orientato principalmente (ma non esclusivamente) alla scrittura di programmi C++ per Kde. In secondo luogo, e molto più importante per non pochi potenziali utenti, KDevelop gira (con la possibile eccezione di alcuni plugin e strumenti secondari) anche su Windows.

L'editor ha più o meno le stesse capacità di quelli di Anjuta: collapsamento del codice, colorazione delle parole chiave di ogni linguaggio, indentazione automatica e integrazione con il debugger, per mostrare immediatamente la riga di codice sorgente corrispondente a ogni errore. Anche in KDevelop basta passare con il cursore sopra la chiamata di una funzione per aprire un link alla sua descrizione. Il codice sorgente di ogni progetto C++ viene caricato e analizzato per costruire un database interno contenente il tipo e altre proprietà di tutte le variabili e classi software.

In generale, KDevelop lavora per progetti e sessioni, che sono gruppi di progetti collegati più o meno strettamente dall'uso delle stesse librerie di base, o da altri vincoli comuni. Il primo caso in cui questa struttura si rivela utile è lo sviluppo simultaneo di due versioni di uno stesso programma. L'altro è quello in cui si lavora su programmi distinti, ma strettamente interdipendenti. Il protagonista di questo secondo scenario potrebbe essere uno sviluppatore che debba scrivere per Linux sia codice del kernel vero e proprio, sia di qualche driver hardware. Poiché qualsiasi driver, per sua stessa natura, deve interagire direttamente col kernel, per farlo dovrà conoscere costanti e lanciare funzioni definite nel codice di quest'ultimo. Di conseguenza, qualsiasi cambiamento nel kernel potrebbe forzare anche una ricompilazione del driver, anche se i due oggetti e i relativi blocchi di codice rimangono ben distinti. In KDevelop questa situazione si gestisce creando un progetto per il kernel, uno per il driver e poi collocandoli all'interno di un'unica sessione.

Gli outsider

Anjuta e KDevelop sono stati sviluppati dai e per i due desktop Linux più popolari del momento, ovvero Gnome e Kde. Per questo sono, almeno in questo periodo, gli ambienti di sviluppo integrato che fanno la parte del leone su questo sistema operativo. Ciò non significa che siano le uniche alternative possibili e nemmeno le migliori per tutti i tipi di problemi. In pratica, l'ambiente di sviluppo migliore per un dato programma dipende da quante persone debbano lavorarci e dal linguaggio in cui si vuole o deve lavorare. Per mancanza di spazio non possiamo sviluppare l'argomento come meriterebbe. Speriamo comunque che le tre Ide presentate nei paragrafi successivi bastino a dare un'idea di quanto sia vasta la scelta Open Source in questo campo.

CodeBlocks (www.codeblocks.org)

L'interfaccia grafica di CodeBlocks, estensibile tramite diversi plugin, gira su Linux, OS X e Windows. CodeBlocks è stato scritto per sviluppare programmi C, C++ e Fortran e può farlo appoggiandosi a diversi compilatori: oltre a quello ufficiale del progetto Gnu, Gcc, sono supportati ufficialmente anche Msvc e Borland C++. Un altro vantaggio di CodeBlocks, anche se in alcuni casi potrebbe causare problemi di compatibilità, è il suo sistema di compilazione personalizzato: molto veloce, anche grazie alla capacità di distribuire il lavoro sui vari core di una stessa Cpu, ma senza Makefile tradizionali. A parte questa peculiarità, CodeBlocks ha molte funzioni in comune con i suoi concorrenti più conosciuti, dall'editor specializzato per la programmazione a una interfaccia completa per il debugger Gnu, Gdb (ma volendo se ne possono anche utilizzare altri). In ogni caso, per breakpoint, controllo in tempo "reale" di tutti i registri del processore, dump completi della memoria ed esecuzione controllata del codice non c'è problema: queste e altre operazioni avanzate sono tutte a portata di clic in CodeBlocks.

Altre funzioni degne di nota sono il generatore di diagrammi della struttura del codice, un'interfaccia grafica per il profiler Gprof e lo strumento per importare progetti creati in altre Ide come Microsoft Visual Studio e DevC++. I file sorgente si possono salvare anche in formato Pdf o Html, per pubblicarli online.

Julia Studio (<http://forio.com/products/julia-studio>)

Julia Studio è probabilmente il prodotto più specializzato di cui parliamo questo mese. In ambito software, Julia è il nome di un linguaggio di programmazione dinamico e ad alte prestazioni, sviluppato soprattutto per il calcolo numerico. Julia Studio, a sua volta, è l'ambiente integrato di sviluppo creato specificamente per scrivere e compilare codice Julia. Il suo workspace contiene l'editor di codice vero e proprio, una barra laterale con l'elenco dei file aperti e delle librerie disponibili, un file manager e la storia dei comandi già eseguiti. Dalla stessa finestra si possono attivare anche una console e vari pannelli in cui visualizzare lo stato delle variabili di un programma, oppure veri e propri grafici. Dietro le quinte, Julia Studio nasconde un compilatore sofisticato, capace di girare in parallelo su più computer per velocizzare il lavoro. I risultati sono programmi di calcolo numerico ad alta precisione, grazie anche a una vasta libreria di funzioni matematiche di ogni tipo, dalla generazione di numeri casuali ad algebra lineare e calcolo delle trasformate di Fourier.

Light Table (www.lighttable.com)

L'annuncio originale di questo programma era un grido di insoddisfazione causato dalla tendenza comune a Ide tradizionali come KDevelop o Anjuta: a ogni versione, diceva l'annuncio, il programmatore si ritrova costretto a lavorare sempre nel solito modo, ma con sempre meno spazio a disposizione, perché è apparsa qualche nuova finestra o menu. Per reazione è nato Light Table, a partire da un concetto molto semplice: un programma di quel tipo non deve essere solo l'integrazione di un editor e di un file manager specializzati: deve soprattutto rendere l'intero processo di sviluppo e validazione software più interattivo e naturale possibile. Light Table cerca di raggiungere questo obiettivo offrendo valutazione e visualizzazione in tempo reale degli effetti di ogni modifica al codice sorgente. In altre parole con Light Table diventa possibile (a certe condizioni) modificare programmi mentre girano al suo interno.

Valgrind

Qualunque siano il sistema operativo, il settore dell'informatica in cui si programma, la ragione per cui lo si fa (studio, lavoro o semplice hobby) e l'ambiente di sviluppo usato, una cosa è certa: non appena si inizia a sviluppare software di complessità medio-alta lo sforzo per eliminare bachi e ottimizzare il codice diventa enormemente più grande di quello passato a scriverlo. O meglio, lo diventerebbe, almeno sotto Linux, senza strumenti come Valgrind (<http://valgrind.org>) la suite Open Source più potente per debugging e profilazione semiautomatici per questo sistema operativo. Il termine profilazione indica tutte le analisi del codice, in parte statiche e in parte fatte mentre sta girando, che consentono di misurare prestazioni, consumo di memoria e altri problemi di ogni componente di un programma.

Valgrind permette di individuare automaticamente molti bug di gestione della memoria e colli di bottiglia, cioè quelle singole funzioni che rallentano moltissimo tutto il resto del programma. I suoi componenti sono stati collaudati con successo anche su applicazioni molto complesse, dell'ordine di decine di *milioni* di linee di codice, in qualsiasi campo della progettazione software.

Valgrind è anche (relativamente) facile da usare: non c'è alcun bisogno di ricompilare o modificare in alcun modo il codice sorgente. Per effettuare tutte le analisi basta far lanciare il programma già compilato da Valgrind stesso, passandoglielo sulla riga di comando. I componenti di Valgrind sono stati inizialmente concepiti per analizzare programmi scritti in C e C++, semplicemente perché quei linguaggi vengono spesso usati per applicazioni complesse, con maggiori possibilità di bug. Allo stesso tempo, la possibilità di far lanciare a Valgrind file eseguibili consente di analizzare programmi in qualsiasi linguaggio, anche quando il loro codice sorgente non è affatto disponibile.

Il prezzo da pagare per un "controllo qualità" così completo e sofisticato è un rallentamento notevole (da cinque a, in casi estremi, anche cento volte!) dei programmi sotto collaudo, quando girano dentro la gabbia virtuale che Valgrind gli costruisce intorno per osservarli. Non è affatto un problema, sia perché Valgrind va usato solo in fase di collaudo, sia perché il tempo passato in attesa che finisca è minore di quello che si perderebbe effettuando le stesse analisi a mano. •

LINUX News

Nuova versione di FreeDomotic per una casa intelligente e Open Source

È un software aperto Java, quindi multiplatforma, per controllare l'automazione della casa o di qualunque altro ambiente (<http://freedomotic.com>). Appartamenti grandi e piccoli, scuole e uffici possono essere gestiti dal pannello di controllo di questo programma, che è anche altamente personalizzabile con diversi plugin. FreeDomotic è stato scritto fin dall'inizio per integrarsi con i sistemi più popolari di domotica, come Bticino, Knx, X10 e Modbus. A partire dalla versione 5.5 FreeDomotic è diventato ancora più flessibile, grazie a diversi cambiamenti dell'architettura che facilitano grandemente sviluppo e integrazione di plugin. Quelli già disponibili includono interfacce per kit Arduino, sensori di alba e tramonto, sintetizzatori e riconoscitori vocali.

Anche il Comune di Piacenza passa a LibreOffice

Entro il 2014 le oltre ottocento postazioni informatiche del Comune di Piacenza adotteranno la suite da ufficio Open Source LibreOffice, con un risparmio complessivo stimato di oltre 250mila Euro. L'iniziativa è partita dopo un'analisi della struttura e degli effettivi bisogni informatici dell'Ente avviata nel 2012. Fondamentale, in quella fase, il questionario somministrato a oltre 500 dipendenti per avere un quadro completo sull'effettivo utilizzo delle singole applicazioni. Ora il programma prevede la formazione di oltre seicento impiegati, che verranno anche assistiti in ogni fase della migrazione da "facilitatori" interni all'amministrazione. L'intera operazione si svolgerà con il supporto del gruppo LibreUmbria (www.libreumbria.it), che ha già affrontato con successo migrazioni dello stesso tipo nella sua regione.

Nuovi container per Linux

I Linux Container (Lxc, <http://linuxcontainers.org>) sono un metodo relativamente recente di creazione e gestione di più macchine virtuali Linux su un solo computer fisico. Lxc 1.0, disponibile dall'inizio dell'anno, è la prima versione stabile a contenere tutti i componenti necessari per creare contenitori senza accesso privilegiato, cioè da amministratore, al sistema ospite. Altre due novità sono il supporto per la clonazione di contenitori già esistenti e la possibilità di salvarne lo stato completo in un generico istante (snapshotting). Fanno parte di Lxc 1.0 anche nuove interfacce di programmazione per i linguaggi Lua, Python 3, Go e Ruby.

Il Ministero dell'Educazione rumeno invita le scuole a valutare Ubuntu

Possibili novità in arrivo per gli studenti rumeni: il loro Ministro dell'Educazione ha pubblicato le nuove linee guida da seguire dopo la recente scadenza di un accordo quadro, non più rinnovato, per la fornitura di software proprietario a tutte le scuole del paese. Il ministro ha esortato tutte le scuole a valutare ed eventualmente aggiornare le loro strutture informatiche, per non farsi trovare con licenze non in regola. La novità è che, oltre a soluzioni irrealistiche, come tornare a versioni precedenti dei vari software, o di fatto impossibili, come comprare nuove licenze con i propri fondi, per la prima volta in Romania è ufficialmente menzionata una terza alternativa: passare a programmi e sistemi Open Source come, per esempio, Linux Ubuntu.