UNIVERSITÀ DEGLI STUDI DELLA CALABRIA

Facoltà di Ingegneria

Corso di Laurea Triennale in Ingegneria Informatica

Tesi di Laurea

TCL PAD: UN EDITOR DI TESTO ESTENDIBILE, INCORPORABILE E CON INTERFACCE DI TEXT-PROCESSING

Relatore:
Prof. SERGIO FLESCA

Laureando:
ANTONIO BONIFATI
MATR. 76515

ANNO ACCADEMICO 2004-2005

In programming, the devil hides in the details.

— Prof. Niklaus Wirth

Introduzione

Esistono centinaia di editor di testo differenti che vengono attivamente sviluppati ed utilizzati, aventi le funzionalità più disparate per praticamente tutti i tipi di sistemi operativi che sono oggi in uso e molti altri ne sono stati usati nel passato.

In Unix i giganti sono l'editor modale Vi (VIsual) di Bill Joy con le sue evoluzioni successive e Emacs (Editor MACroS) di Richard Stallman, che sono molto potenti, ma anche molto difficili da usare.

Emacs in particolare si è evoluto fino a diventare molto più di un editor: tra le altre cose funziona anche da programma di posta elettronica, da shell, da browser e può essere usato per compilare programmi e persino per la psicoterapia automatizzata (vale da dire che fa il dottore :-). È facilmente estendibile perché i suoi comandi di editing sono scritti in Lisp.

Quale posto ricopre allora il neonato tpad di cui tratta questa tesi nell'universo degli editor?

Strettamente parlando tpad non è proprio un editor, ma piuttosto una libreria di funzioni di alto livello per la creazione di editor di testo e, in futuro, anche di programmi di word-processing più sofisticati. Con tpad viene distribuito anche un programma di esempio che, facendo uso dei comandi esportati dalla libreria, implementa un semplice editor tipo SimpleText sul Mac o Notepad di Windows, con l'aggiunta di alcune nuove funzioni che avevo sempre desiderato. Questo editor, a causa di una temporanea mancanza di fantasia, si chiama tpad, proprio come la libreria e funge anche da test per la stessa.

Normalmente le librerie grafiche mettono a disposizione dei *widget* in grado di rappresentare del testo semplice o formattato. Tk, il toolkit per le GUI di Tcl, probabilmente la più semplice da usare di tutte le librerie grafiche in circolazione, non fa eccezione. Il *widget text* tuttavia rappresenta solo un punto di partenza. Se si vuole creare un vero e proprio editor occorre ancora molto altro lavoro per implementarne i comandi.

Ci sono molti programmi che, pur non essendo un word-processor, hanno bisogno di un editor di testo integrato: i più comuni sono i debugger, gli editor HTML o XML, i programmi di posta elettronica, gli ambienti integrati di sviluppo (IDE) e in generale tutte le applicazioni in cui l'utente ha la necessità di editare delle stringhe lunghe. Persino programmi con cui sembrerebbe che l'editing abbia poco a che fare, come le interfacce sui database o gli applicativi per il disegno di diagrammi potrebbero beneficiare dalla presenza di qualche funzione di editing avanzata, in quanto l'utente potrebbe avere la necessità di gestire talvolta testi più lunghi di una riga. Per questo l'editor può essere considerato un'applicazione ubiqua.

Sembra che ciascun programma che abbia visto, a partire da qualche tipo di widget, abbia implementato per suo conto l'editor integrato con le funzioni di cui ha bisogno. Mi sono allora chiesto se si potesse fare il lavoro una volta per tutte, applicando quel vecchio principio dell'ingegneria del software che si chiama riusabilità, piuttosto che reinventare la ruota ogni volta.

Il risultato è tpad o Tcl pad che, dalle mie ricerche, risulta l'unico che è contemporaneamente estendibile, si può incorporare in altre applicazioni in vari modi, è portabile a livello di sorgente, configurabile in ogni dettaglio, facilmente comprensibile e di piccole dimensioni.

Tpad deve molte delle sue buone proprietà al linguaggio Tcl/Tk con cui è interamente scritto e che serve anche come linguaggio per le estensioni, proprio come avviene per il Lisp in Emacs.

Tpad cerca di fare solo l'editor, non l'ambiente di lavoro. Tuttavia è dotato di una semplice architettura per la creazione di plugin che permettono sia di sfruttare altre applicazioni esterne, semplificandone l'uso, sia di implementare rapidamente nuove funzionalità direttamente in Tcl/Tk [§ 1.4, p. 8]. Si possono creare plugin di qualsiasi tipo, ma chiaramente l'interesse maggiore è rivolto verso tutte quelle applicazioni che hanno a che fare con l'elaborazione del testo: preprocessori, compilatori, convertitori, correttori, validatori, strumenti di analisi dei testi e del codice dei programmi, sistemi per la preparazione dei documenti basati sui linguaggi, formattatori, dizionari e molti altri programmi speciali.

Dato che le estensioni vengono caricate dinamicamente solo quando vengono effettivamente utilizzate, è possibile anche averne installate a centinaia senza che questo influisca sulle risorse di calcolo utilizzate dall'editor.

Quali sono i vantaggi principali che si ottengono nell'utilizzare tpad piuttosto che un altro editor?

Dal punto di vista dell'utente, oltre alla semplicità di utilizzo, è un vantaggio ritrovare lo stesso editor in più applicazioni: in questo modo l'utente deve imparare ad usare solo un tipo di editor; spostandosi da una applicazione all'altra occorrerà imparare solo i pochi comandi specifici aggiunti dall'applicazione. Inoltre i plugin distribuiti insieme alla libreria potranno essere utilizzati da qualsiasi programma che incorpori tpad e quest'ultimo può anche aggiungerne dei propri.

Dal punto di vista del programmatore i vantaggi sono più o meno gli stessi che hanno decretato il successo del sistema operativo Unix, al cui modello di sviluppo tpad si ispira: tpad è piccolo e semplice da capire, ha uno scopo ben definito, è un componente riusabile e ben documentato [Capitoli 2 e 3], ha una interfaccia di libreria semplice e flessibile e non pone limiti predefiniti sulle dimensioni (compresa quella del testo). Tpad può essere riutilizzato in tre modi, a seconda del tipo di controllo desiderato: tramite l'interfaccia a riga di comando [§ 2.4, p. 40], tramite la *Application Programming Interface* (API) della sua libreria [§ 3.1.2, p. 50] o tramite comandi Tcl trasmessi da un'altra applicazione [§ 1.6, p. 17].

Indice

Elenco delle figure

Capitolo 1

Descrizione generale

1.1 Introduzione e caratteristiche principali

Tcl pad (abbreviato in *tpad* e pronunciato come *tea pad*) è un editor di testo, facile da usare, portabile, estendibile con nuove funzionalità, incorporabile in altre applicazioni e munito di front-end grafici sulle più comuni interfacce di text processing.

Tpad è stato scritto interamente nel linguaggio Tcl/Tk ed è distribuito in formato di script sorgenti in formato portabile ASCII a 7-bit. In aggiunta alla versione portabile, sono anche disponibili versioni pacchettizzate per i più comuni sistemi operativi, che automatizzano le operazioni di installazione, disinstallazione e configurazione iniziale [Appendice B].

A differenza della maggior parte degli editor che sono per lo più programmi a sé stanti, tpad è invece una libreria di funzioni di editing [§ A.1, p. 55]. Funzionalità e interfacce che le richiamano sono state tenute separate per massimizzare la riusabilità del codice. Tramite una API molto semplice basata su Tcl nel ruolo di linguaggio "colla" [§ 1.3, p. 4], il codice di tpad può essere riutilizzato per costruire rapidamente editor integrati in altri programmi, come ad esempio debugger orientati allo schermo, ambienti di sviluppo integrati e programmi di posta elettronica.

Tpad può essere usato come editor esterno, operante in una finestra separata da quella dell'applicazione che lo usa (finestra che può essere sia modale che non), oppure può essere facilmente integrato nella applicazione stessa, operando nella sua stessa finestra principale. Per scambiare il testo che deve essere editato tra tpad e l'applicazione, è possibile scegliere tra l'uso di file temporanei e l'editing diretto di una variabile di memoria dell'applicazione stessa (stringa lunga di Tcl) [§ 3.1.2, p. 50].

Tutti i menu, le combinazioni di tasti attive, le funzioni di risposta alle

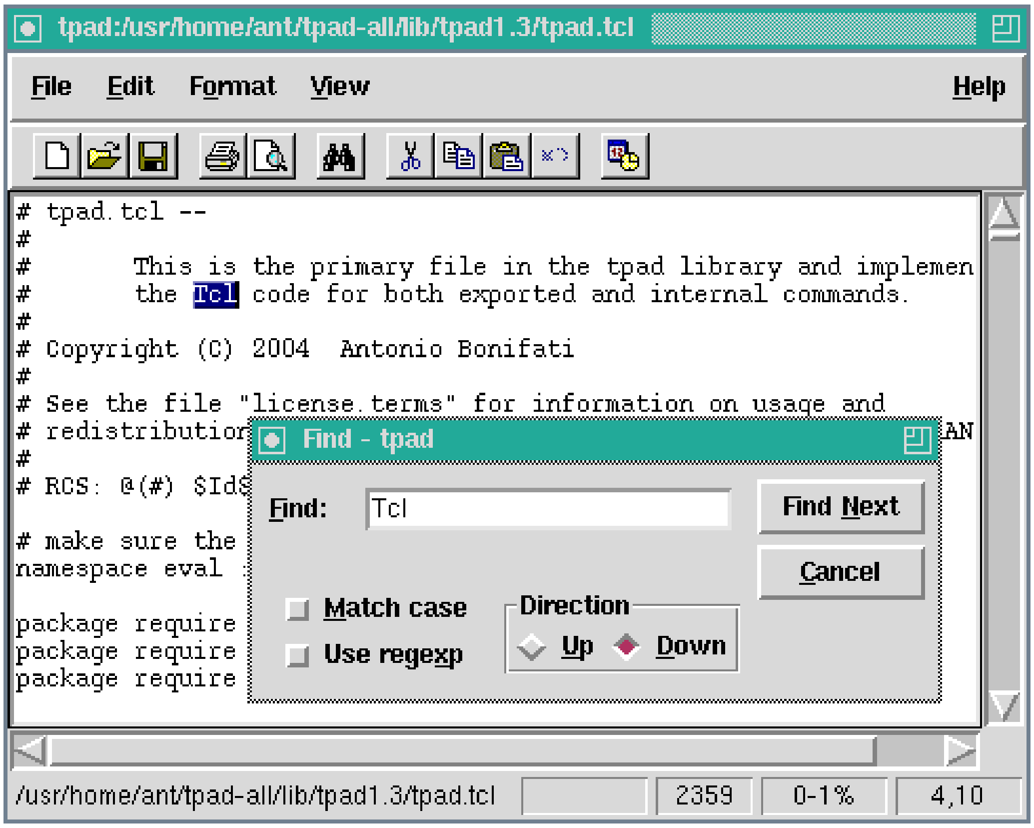

Figura 1.1: Una schermata dell'interfaccia di default di tpad

interazioni dell'utente con l'interfaccia grafica *(callback functions)*, il modo in cui gli elementi dell'interfaccia grafica (detti *widget*, abbreviazione di *window object*) sono combinati tra loro possono essere ridefinite, qualora necessario, editando un unico file di codice Tcl di dimensioni contenute. Nella versione corrente questo file consta di meno di 400 linee [Appendice A.2], avendo incluso nel conteggio anche le numerose linee vuote e di commento presenti.

Diverse istanze di tpad possono essere attive contemporaneamente, in modo indipendente tra loro. All'atto della creazione una istanza può essere configurata per agire come editor oppure come solo visualizzatore di testo. Una istanza di tipo viewer ha una interfaccia simile ad una di tipo editor, ma non sono presenti tutte quelle funzionalità che comportano la modifica del contenuto del testo [§ 2.4.1, p. 42].

Nella definizione dell'interfaccia grafica di default e nella scelta delle funzioni di editing fondamentali, ho deciso di rendere tpad molto simile al ben noto Blocco Note di Microsoft Windows XP (detto Notepad nell'edizione in lingua inglese). I menu sono organizzati alla stessa maniera, le combinazioni di tasti [§ 2.2, p. 35] sono le stesse, e la toolbar è una copia addirittura a livello di pixel di Wordpad, non essendo presente nel Notepad originale [Figure 1.1 e 1.2]. Per questa somiglianza esteriore, tpad può essere definito brevemente come un clone "avanzato" di Notepad.

tpad è un progetto open-source soggetto alla Licenza Generale Pubblica GNU [1] che viene gestito tramite il portale collaborativo per lo sviluppo software Sourceforge.net. È inoltre entrato a far parte della collezione delle port del sistema operativo FreeBSD [2] e viene distribuito come pacchetto contrib RPM dalla distribuzione Linux Mandriva (precedentemente nota come Mandrake).

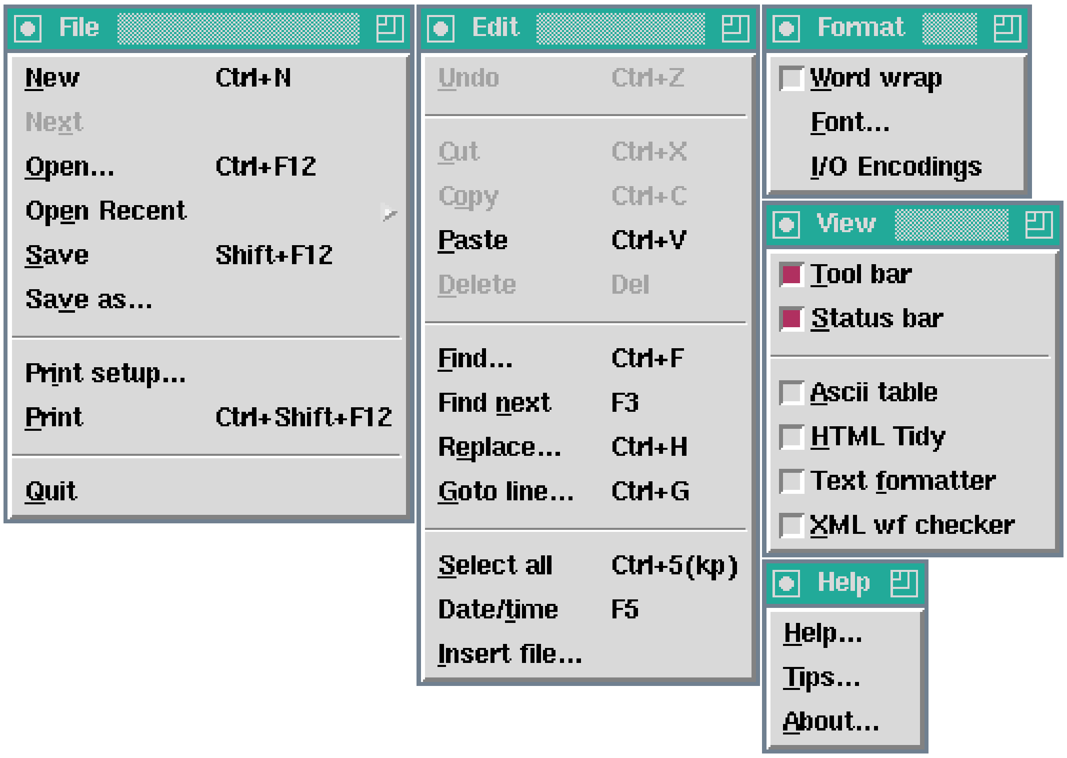

Figura 1.2: I menu staccabili contengono tutte le funzioni implementate

1.2 Internazionalizzazione e localizzazione

Tpad è una applicazione pienamente I18N/L10N-compliant. Tutti i menu e i messaggi nei dialoghi possono essere facilmente tradotti a runtime e il programma è in grado di ricevere in input ed elaborare del testo in tutte le principali lingue del mondo.

Inizialmente ho scritto i cataloghi dei messaggi nelle sole due lingue che conosco, l'Inglese e l'Italiano. Poi con l'aiuto di altri collaboratori madrelingua, è stata aggiunta la traduzione in Francese, Spagnolo, Tedesco ed Ebraico. Sto cercando attualmente altri traduttori. Per essere davvero internazionale, tpad dovrebbe supportare almeno tutte le maggiori lingue del mondo, incluse quelle ideografiche come Cinese, Giapponese e Coreano e naturalmente anche il Russo.

Per agevolare la diffusione internazionale del programma, la lingua base di tpad è l'Inglese. In questa lingua ho scritto tutta la documentazione allegata al programma (che si divide in documentazione per gli utenti e documentazione per i programmatori), i numerosi commenti che fanno parte del codice e le pagine web del sito ufficiale. L'Inglese è anche la lingua che uso per comunicare, prevalentemente via email, con i traduttori. Nei [Capitoli 2 e 3] si trova la maggior parte della documentazione online appositamente tradotta in italiano per questa tesi.

Attualmente mantengo sincronizzati tramite il protocollo di aggiornamento remoto rsync [3], tre mirror completi del sito web di tpad, disposti fisicamente in Italia, negli USA e in India.

I contenuti principali del sito [4] sono:

- breve descrizione del progetto e screenshot del programma
- informazioni sulle versioni disponibili
- collegamenti per scaricare il software e dettagli su come installarlo
- indicazioni per i contributor
- patch per eventuali bug scoperti dopo il rilascio della versione stabile

Per ottenere l'ultima versione di sviluppo (ovvero instabile) del programma è disponibile un servizio di CVS anonimo messo a disposizione da Sourceforge.net.

1.3 Il linguaggio Tcl/Tk

Tcl [5] (Tool Command Language, pronunciato come *tickle*) è un linguaggio di comandi ad alto livello, non proprietario, interpretato, facilmente estendibile, dinamico, portabile e incorporabile. Appartiene alla categoria dei linguaggi di scripting [6] non tipati piuttosto che a quelli di sistema e ha subito influenze da altri linguaggi quali C, Lisp e C-shell.

Una sintassi molto semplice e uniforme, basata solo su comandi (detti anche procedure) aventi parametri con nome, rende Tcl molto facile da usare e imparare. Le strutture di controllo, le definizioni di nuove procedure sono esse stesse dei comandi e non vi sono operatori a livello sintattico. Per la valutazione delle espressioni si usa infatti un apposito comando *expr* in grado di interpretarle come parametri. Ad un comando possono essere forniti altri comandi come argomenti, sotto forma di stringhe, che il comando tipicamente passerà all'interprete Tcl per ottenerne l'esecuzione.

A scopo illustrativo, segue un semplice esempio di applicazione Tcl/Tk:

```
button .b -text Hello! -font {Times 16} -command {puts hello}
pack .b
```

Questa è una banale applicazione Hello World! grafica, che consta di un unico tasto la cui pressione provoca la stampa della stringa `hello` sul terminale. Il creatore di Tcl, John K. Ousterhout, mostra [6] come lo stesso esempio implementato in Java richiederebbe 7 linee di codice in due metodi differenti (senza contare le altre 21 linee di codice necessarie per creare un *applet*) e ben 25 linee di codice in tre procedure utilizzando C++ con le Microsoft Foundation Classes (avendo escluso ancora una volta la corposa infrastruttura di codice necessaria per qualsiasi applicazione C++/MFC).

Tcl incorpora già molte delle funzionalità utili in un linguaggio di scripting moderno e avanzato. La lista seguente è sicuramente incompleta, ma rende l'idea delle sue potenzialità:

- comandi per operare sulle stringhe, le liste e gli array associativi
- supporto per espressioni regolari avanzate
- namespace dinamici per creare contesti separati per comandi e variabili
- package per facilitare il riuso, la modularizzazione e la distribuzione del codice
- dynamic loading [1]
- tracing delle variabili e dei comandi
- supporto per la programmazione event-driven, socket e thread [2]

[1] Permette di caricare estensioni su richiesta a run-time, riducendo i tempi di startup delle applicazioni.

[2] Tcl è thread-safe; nessun comando Tcl crea conflitti quando vi sono più thread concorrenti in esecuzione.

- error trapping
- comandi portabili per la gestione del file-system e delle porte seriali
- funzionalità di internazionalizzazione e supporto Unicode [7]

I comandi predefiniti supportano solo tre tipi di strutture dati (stringhe, array associativi e liste) e un solo tipo di dati, le stringhe, che sono sufficienti per molte applicazioni. Tuttavia, mediante opportune estensioni, molte delle quali già disponibili e pronte all'uso, si possono aggiungere altri tipi e strutture dati in maniera efficiente e senza complicare la sintassi di base di Tcl. Ad esempio l'estensione BLT mette a disposizione il tipo dati vector (vettori di valori floating point), importanti per molte applicazioni di calcolo.

Tcl puro è ideale per scrivere velocemente piccole applicazioni o prototipi. Se combinato con il toolkit Tk [8] (pronunciato come *tik*) e altre estensioni se necessario, può essere usato con successo anche per sviluppare applicazioni di media/grande dimensione e dotate di interfaccia grafica. Le ultime versioni di Tcl hanno tutte le funzionalità richieste per creare applicazioni aziendali mission-critical. Tramite estensioni è supportato l'accesso ai più popolari sistemi di gestione di basi di dati, come Sybase, Oracle e MySQL.

Tk è una estensione di Tcl che rende molto semplice creare sistemi interattivi perchè mette già a disposizione i componenti riusabili necessari; come visto in piccolo nel semplice esempio del programma "hello", questi vengono poi assemblati e integrati per costruire l'interfaccia grafica tramite l'uso del linguaggio Tcl. Le GUI create con Tk sono portabili tra Unix, Macintosh, Windows e assumono l'aspetto e i comportamenti dell'interfaccia di default del sistema su cui girano, che è quella a cui l'utente è maggiormente abituato.

Da uno script Tcl è possibile eseguire altri programmi esterni, redirigendone l'input e/o l'output, come si fa negli script di shell Unix. Esistono due shell tipo Unix contenenti l'interprete Tcl: tclsh e wish. Tclsh è una shell che utilizza come comandi interni quelli di Tcl e come esterni quelli messi a disposizione dal sistema operativo ospite. Wish sta per *WIndowing SHell* e, rispetto a tclsh, è dotata di finestra grafica e aggiunge i comandi Tk. Con wish gli amministratori di sistema possono crearsi velocemente script dotati di interfacce grafiche sui comandi Unix, che consentono di automatizzare e semplificare le operazioni ripetitive [9].

Per la distribuzione di applicazioni non open-source in forma solo compilata, è possibile ricorrere ad un compilatore Tcl [10] che genera bytecode portabile (codificato in formato testuale ASCII a 7 bit). Questo non è però al momento ancora disponibile per la piattaforma Macintosh.

Tcl è anche un linguaggio di scripting universale per Internet. Può essere usato come linguaggio di script lato client embedded in HTML, similmente a JavaScript o come plugin esterno del browser, per creare appletti (detti tclet) analoghi agli applet Java. Come Java, consente di eseguire in tutta sicurezza gli script scaricati, grazie ad un sofisticato meccanismo di controllo e assegnazione dei privilegi detto Safe-Tcl. Esiste poi Tclhttpd, che è una implementazione in Tcl puro e con buone performance di un server web basato sul protocollo HTTP e dotato di un framework per la generazione di pagine con contenuto dinamico. Poichè Tcl è un linguaggio facilmente incorporabile in altre applicazioni, anche Tclhttpd può essere incorporato, ad esempio allo scopo di aggiungere rapidamente un'interfaccia web alle vecchie applicazioni legacy.

L'interprete Tcl è strutturato come una libreria di funzioni che può essere incorporata in altri linguaggi, sia di sistema (C, C++, Java) sia di scripting (Python). L'interprete è disponibile con licenza open-source sulle tre piattaforme più diffuse (Unix, Macintosh, Windows).

Supponiamo, per esempio, di voler sviluppare un'applicazione di foglio elettronico in uno dei suddetti linguaggi di sistema: anziché inventare un altro linguaggio per le formule e la definizione delle macro funzioni, si può usare Tcl come linguaggio incorporato. Così facendo non è necessario definire un nuovo linguaggio e scrivere il corrispondente parser e interprete in quanto questi vengono già forniti con poco sforzo includendo la libreria Tcl. Tutte le funzionalità generiche di un linguaggio di comandi, come le strutture di controllo, le variabili, le procedure sono già fornite da Tcl: occorre solo estendere i comandi di base Tcl con i comandi addizionali necessari alla particolare applicazione che si sta scrivendo. Questi nuovi comandi possono essere scritti in C, C++, Java o in Tcl stesso e facilmente incorporati grazie alla semplice interfaccia fornita dal parser Tcl. L'aggiunta di comandi specifici di un'applicazione arricchisce il linguaggio Tcl senza variarne minimamente la sintassi; non c'è alcuna differenza tra i comandi predefiniti di Tcl e quelli aggiunti dall'applicazione.

È possibile creare estensioni (ovvero set di nuovi comandi) che lavorano con diverse versioni di Tcl senza necessità di ricompilazione.

Alcune estensioni Tcl general-purpose e liberamente disponibili:

incr Tcl Estensione che aggiunge complete funzionalità object-oriented a Tcl, ispirata al linguaggio C++.

Expect Estensione che consente di automatizzare il dialogo con i programmi interattivi in Unix.

TclX Detto anche Extended Tcl, set di estensioni orientato principalmente alla programmazione di sistema di Unix.

Tcllib Libreria di codice di utilità scritto in Tcl puro. Attualmente consta di circa 80 moduli. Ogni modulo contiene un certo numero di comandi correlati.

Snack Toolkit per la manipolazione e l'analisi dei suoni, per creare applicazioni audio multi-piattaforma.

BLT Toolkit che aggiunge nuovi tipi di widget più complessi, tra cui grafici 2D, controlli ad albero e contenitori a schede.

Attualmente tpad utilizza solo TclX e Tcllib, per alcune funzioni avanzate, tra cui il locking dei file e la creazione dei file temporanei in modo portabile. Tpad può essere utilizzato anche se queste estensioni non sono installate. Naturalmente in questo caso le funzionalità che richiedono queste librerie aggiuntive non saranno disponibili.

1.4 I plugin di tpad

Tpad può essere facilmente esteso tramite l'aggiunta di plugin specifici, che permettono di invocare utility esterne per l'elaborazione del testo senza lasciare l'editor.

La scrittura di un nuovo plugin non richiede una conoscenza dettagliata del resto del codice di tpad, ma solo l'adesione ad una serie di semplici convenzioni necessarie affinché il plugin possa essere invocato dall'interno di tpad ed operare correttamente la lettura ed eventuale modifica del testo dell'editor.

Ogni plugin è "specifico" nel senso che "conosce" l'interfaccia a riga di comando, il modo di operare e il formato dei messaggi di errore emessi dall'applicazione esterna che esegue.

I plugin forniscono all'utente una interfaccia grafica facile da usare con le seguenti funzionalità (ove applicabili):

- impostazione delle opzioni del programma da richiamare in modo consistente, prevenendo ogni tipo di errore di invocazione
- raccolta di eventuali messaggi di errore forniti dal programma esterno, per consentirne la comoda consultazione da parte dell'utente
- opzione per stabilire se il filtro esterno deve essere applicato all'intero testo o solo ad una sua porzione selezionata
- posizionamento veloce del cursore di inserzione dell'editor alla riga e alla colonna del testo dove si è verificato un errore

I plugin vengono caricati dinamicamente la prima volta che vengono visualizzati, tramite la funzionalità di auto-loading del codice di Tcl. In questo modo può essere sviluppato anche un numero elevato di plugin, senza che i tempi di startup dell'applicazione si allunghino.

Ho a lungo pensato anche ad un tipo di plugin generico, che possa essere configurato tramite GUI, per generare nuovi plugin in grado di richiamare qualsiasi tipo di filtro o applicazione esterna. Tuttavia mi sono reso conto che per generare plugin specifici con tutte le caratteristiche volute, il plugin generico dovrebbe essere molto complicato sia da implementare sia da configurare e di certo non potrà mai avere la flessibilità consentita dalla programmazione diretta in Tcl di un nuovo plugin. Perciò qualora decidessi di includere in futuro un plugin generico che permetta anche agli utenti non programmatori di creare dei semplici plugin, questo non sarà sostitutivo dei plugin specifici già implementati.

L'[Appendice A.3, p. 78] contiene il codice completo di uno dei plugin.

1.4.1 Plugin Tabella ASCII

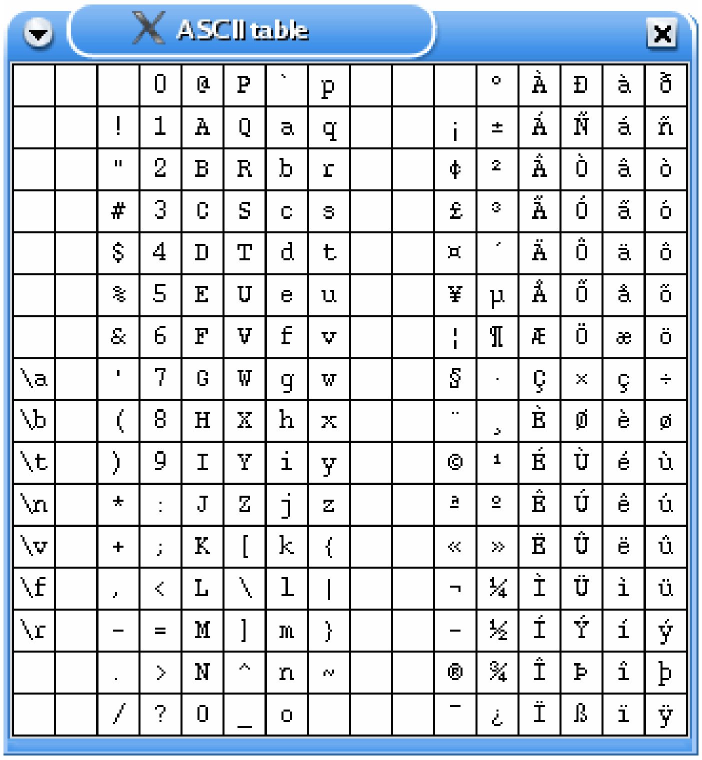

Figura 1.3: Il plugin Tabella ASCII

Questo plugin [Figura 1.3] visualizza una griglia quadrata contenente 256 caselle (16x16): nelle caselle sono rappresentati i caratteri del set ASCII standard (codici da 0 a 127) e quelli dell'estensione nota come Latin-1 (codici da 160 a 255). Di solito i font non comprendono i glifi corrispondenti ai caratteri di controllo (i primi 32 del set ASCII standard) e quelli non appartenenti all'estensione Latin-1. In questo caso le caselle corrispondenti sono vuote, eccetto per alcuni caratteri di controllo che vengono rappresentati con le ben note sequenze di escape usate in C o Tcl. Cliccando con un tasto del mouse su un carattere, questo viene inserito nel testo, alla posizione corrente del cursore.

La tabella ASCII risulta utile per inserire nel testo caratteri latini di cui manca il tasto apposito sulla tastiera.

1.4.2 Plugin Correttore e validatore di HTML

HTML Tidy [11] è un validatore, correttore e formattatore di codice HTML, scritto da Dave Ragget del W3C.

Tidy è in grado di aggiustare automaticamente molti errori presenti nel markup e, nei casi di ambiguità, emette messaggi di errore esplicativi che aiutano il webmaster a correggere manualmente il problema [§ 2.5, p. 2.5]. Tidy migliora molto il pessimo markup generato da alcuni editor HTML e strumenti di conversione. Ha perfino una modalità apposita per ristrutturare l'orribile markup generato da Microsoft Word 2000! Inoltre fornisce una serie di consigli per rendere le pagine più accessibili a persone disabili.

Tidy è un programma molto ricco e può essere configurato con circa una cinquantina di opzioni. Allo scopo di rendere l'interfaccia grafica semplice ho selezionato le sole opzioni più comunemente usate [Figura 1.4]. Eventuali direttive più sofisticate possono comunque essere specificate, utilizzando un file di configurazione testuale esterno, che può essere modificato lanciando un'altra istanza di tpad.

1.4.3 Plugin Controllore di XML Ben-Formato

Xmlwf è un plugin che determina se un documento XML è ben formato [Figura 1.5 e § A.3, p. 78]. Xmlwf è basato sulla libreria Expat, che implementa in linguaggio C un parser per XML versione 1.0, pienamente conforme alle specifiche W3C.

Xmlwf è un processore non validante: se il documento ha una DTD, non controlla se la sintassi della DTD è corretta e se il documento è conforme alla DTD stessa. Tuttavia supporta il parsing delle entità esterne.

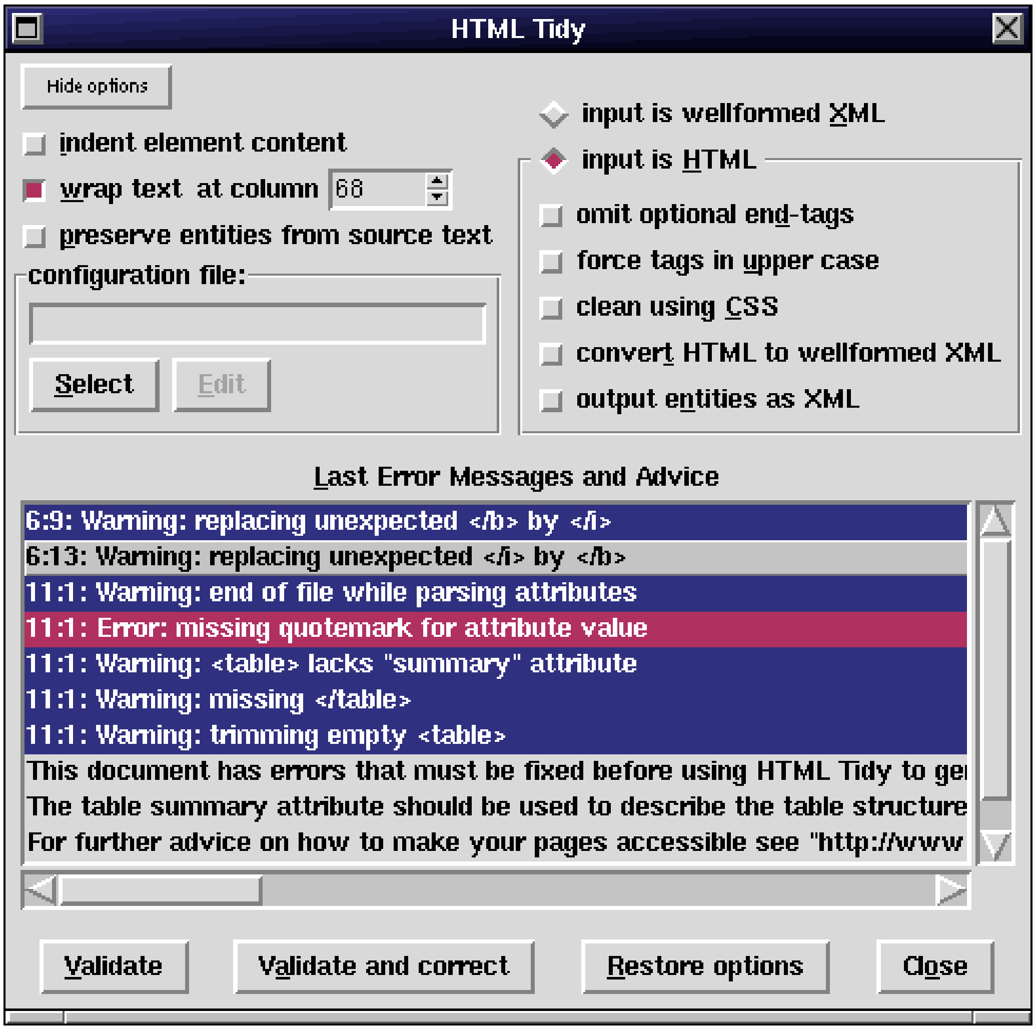

Figura 1.4: Il plugin di HTML Tidy

1.4.4 Plugin Formattatore di Testo

Fmt è il classico filtro formattatore di testo di Unix. Ne esistono fondamentalmente due varianti, che hanno interfaccia e opzioni diverse: quella BSD [Figura 1.6], disponibile per default in FreeBSD e quella GNU [Figura 1.7], tipica di Linux: tpad le supporta entrambe.

La funzionalità fondamentale, comune sia alla versione BSD che GNU, consiste nel riorganizzare il testo, riempiendo quanto più possibile le linee di uno stesso paragrafo, senza superare un certo numero prefissato di caratteri per linea. Per fare ciò le linee vengono unite o spezzate solo in corrispondenza degli spazi bianchi; vale a dire che le parole non vengono mai unite o sillabate.

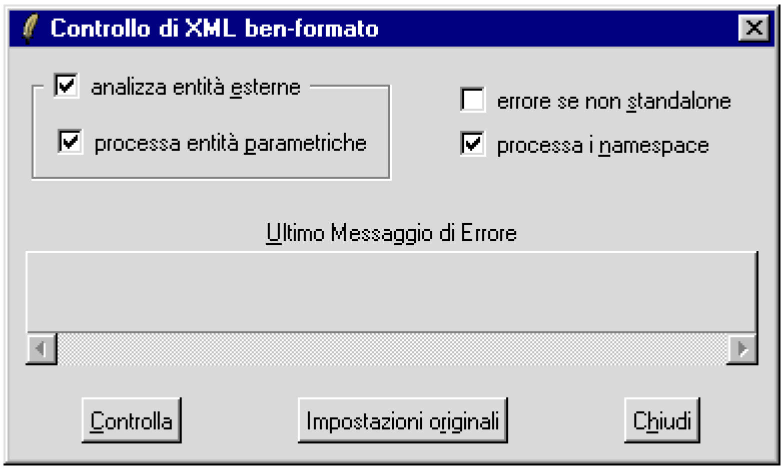

Figura 1.5: Il plugin xmlwf

1.5 Struttura del codice

Il codice di tpad segue le convenzioni dettate dalla Guida di Stile Tcl [12], che è stata elaborata dal gruppo di lavoro su Tcl/Tk dei Laboratori Sun. Le convenzioni di stile usate non sono importanti di per sè quanto lo è il fatto di essere consistenti con esse. L'uso di uno stile uniforme infatti facilita il riuso, la lettura e la manutenzione del codice da parte di un altro programmatore.

Nonostante il nome, la Guida di Stile Tcl non si limita a prescrivere delle regole di stile di codifica. Vi si trovano anche regole pratiche e suggerimenti che riguardano alcuni aspetti importanti per la produzione di software di qualità:

- documentazione e commenti
- scelta appropriata dei nomi degli identificatori
- portabilità
- automatizzazione e riproducibilità dei test
- strutturazione del codice in entità facilmente riusabili e ridistribuibili
- prevenzione degli errori di programmazione

La [Figura 1.8, p. 15] rappresenta il nesting degli spazi dei nomi quando sono attive n istanze di tpad.

Ciascuna istanza è identificata tramite il percorso Tk della finestra grafica che la contiene. I percorsi in Tk hanno sintassi analoga a quelli dei file o delle directory in un filesystem Unix. Le uniche differenze sono che al posto del

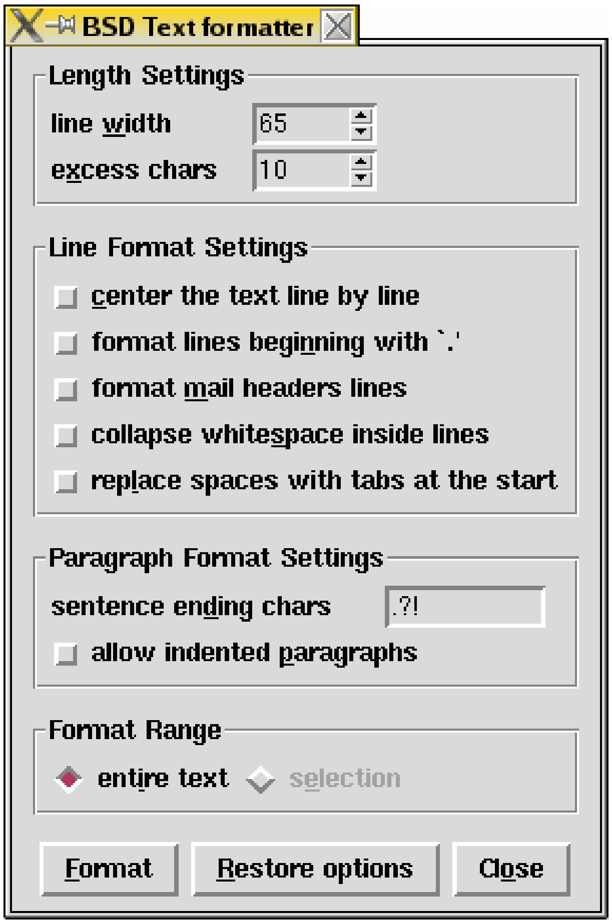

Figura 1.6: Il plugin BSD fmt

carattere slash (`/`) si usa il punto (`.`) e non si può avere un punto alla fine di un percorso, eccezion fatta per il percorso radice (`.`), che rappresenta la finestra principale dell'applicazione. Le n istanze sono state chiamate con i nomi `.edit1 ... .editn` solo per semplicità; in realtà ciascuna istanza può essere creata associandogli un percorso identificatore qualsiasi, purché esista il widget padre.

Tutti i comandi e le variabili di tpad risiedono all'interno del namespace `tpad`; nessuna procedura o variabile di tpad risiede nello spazio nomi globale (`::`). Questa è l'organizzazione raccomandata dalla Tcl Style Guide [12] per evitare il problema della collisione dei nomi, che è il motivo principale per cui sono stati introdotti i namespace: un comando non dovrebbe mai creare o modificare altri comandi o variabili nello spazio globale a meno che non gli venga chiesto esplicitamente tramite la sua interfaccia dei parametri.

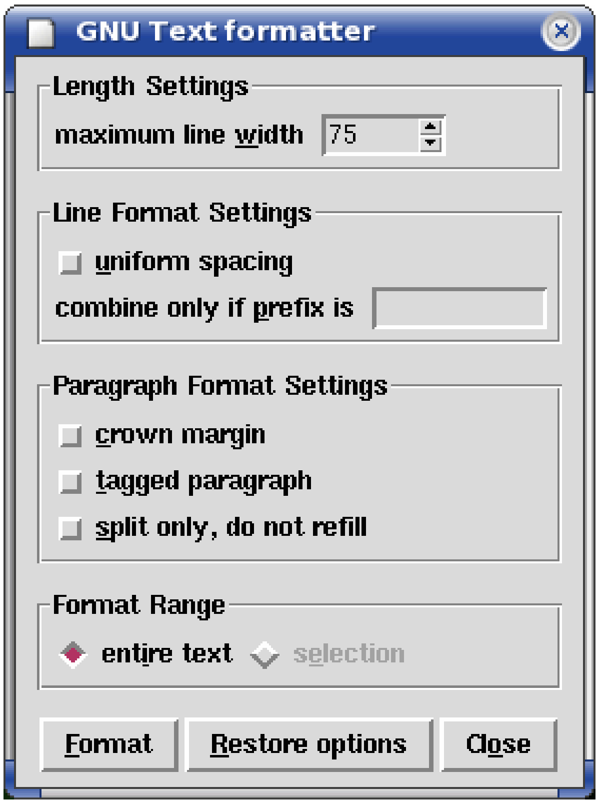

Figura 1.7: Il plugin GNU fmt

I comandi esportati dal namespace `::tpad` costituiscono l'interfaccia pubblica della libreria, e si possono suddividere in tre categorie [§ A.1]:

- comandi di inizializzazione, per creare l'istanza e per visualizzarla
- comandi corrispondenti a tutte le voci del menu dell'interfaccia di default [Figura 1.2, p. 3]
- comandi addizionali, non corrispondenti alle voci del menu

I comandi addizionali sono comandi di utilità che, al pari dei comandi di menu, possono essere invocati dall'esterno in qualsiasi ordine e con qualsiasi argomento, senza causare errori imprevisti oppure perdita di consistenza dello stato di una istanza.

Ad esempio un comando addizionale è `goToLine` che permette di riposizionare il cursore di inserzione ad una certa riga o colonna passata come parametro e riaggiustare la vista nella finestra dell'editor in modo che il cursore sia visibile nella sua nuova posizione. Questo non è il comando che corrisponde

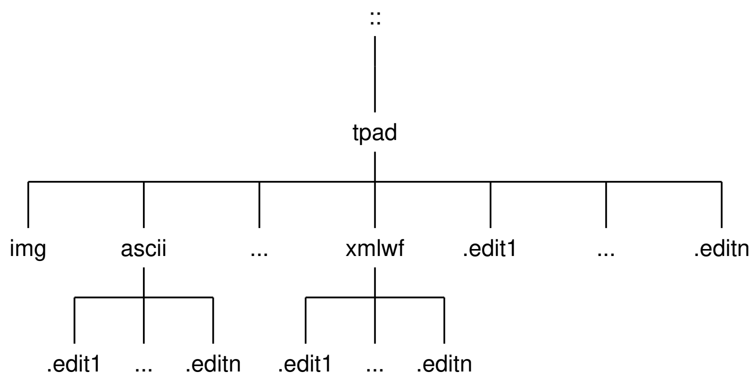

Figura 1.8: La struttura ad albero dei namespace creati da tpad

alla voce del menu "Edit/Goto line", che si chiama invece `goTo`. Quest'ultimo visualizza la finestra di dialogo modale "Goto line" che internamente usa il precedente comando `goToLine`, associandolo alla pressione del tasto *Go*, il tasto che appunto causa il riposizionamento del cursore.

Altri comandi addizionali ritornano informazioni di stato dell'editor, come per esempio il comando `lines`, che ritorna il numero di linee del testo corrente.

Per contro i comandi il cui nome inizia con una lettera maiuscola non fanno parte dell'interfaccia della libreria e sono da considerarsi privati; essi non sono ufficialmente supportati e potrebbero cambiare oppure scomparire senza notifica. Per questo motivo il loro riuso dovrebbe essere evitato.

Quasi tutti i comandi operano su una specifica istanza dell'editor e richiedono come primo parametro il pathname Tk che la identifica.

Il namespace `::tpad::img` [Figura 1.8] non contiene comandi, ma solo nomi di immagini, che funzionano da identificatori nei buffer tenuti da Tk. Le uniche immagini usate fino ad adesso da tpad sono quelle della toolbar e le icone predefinite di Tk (usate nelle finestre di dialogo). Gli ID delle immagini sono tenuti in un namespace separato per amor di ordine all'interno della libreria e per evitare collisioni con altri nomi di immagini eventualmente caricate da un'applicazione che usa la libreria tpad.

Inoltre per ogni istanza dell'editor esiste un namespace innestato all'interno di `::tpad` avente lo stesso nome dell'identificatore dell'istanza. Nell'esempio di [Figura 1.8], questi namespace sono `::tpad::.edit1 ...  ::tpad::.editn`. Ciascuno di questi namespace contiene tutte le variabili di stato dell'istanza come ad esempio il nome del file corrente che si sta editando, il tipo di istanza (se editor o viewer), l'ultimo pattern utilizzato per la ricerca (allo scopo di poterla ripetere), ecc.

Infine ogni plugin vive nel proprio namespace, che contiene sia i comandi del plugin (esportati e privati) sia le eventuali variabili di stato relative al plugin, suddivise all'interno di namespace con nome coincidente con il pathname dell'istanza, secondo la stessa convenzione usata nel namespace principale `::tpad` [Figura 1.8]. Questo assicura che non vi siano mai collisioni dei nomi tra i plugin. Ogni plugin può accedere anche alle variabili dell'istanza dell'editor o ai comandi pubblici o privati della API di tpad, usando i loro nomi pienamente qualificati.

Tpad è un programma completamente configurabile: al momento le direttive di configurazioni manipolabili sono all'incirca 80. La configurazione è memorizzata in un unico e semplice file Tcl, che viene caricato nel momento in cui si crea la prima istanza [§ 2.1, p. 23]. In questo modo non è stato necessario definire un nuovo formato e creare un parser per il file di configurazione. Quando il programma viene installato viene creato un file di configurazione globale, valido per default per tutti gli utenti di un sistema. Gli amministratori di sistema possono cambiare il file di configurazione globale se desiderano imporre diversi valori di default per alcune opzioni. Secondo un meccanismo comune in Unix, ma applicabile anche in altri sistemi multi-utente, gli utenti possono copiare il config file globale in un file nascosto all'interno della loro home directory qualora desiderino partire con valori di default differenti da quelli stabiliti dall'amministratore.

Le direttive di configurazione per tpad si dividono in due categorie:

- variabili o costanti di configurazione che si applicano a tutte le istanze di tpad, create nel namespace `::tpad`

- valori iniziali delle variabili relative ad ogni singola istanza di tpad; queste pure sono definite nel namespace `::tpad` ma vengono poi copiate nei namespace che conservano lo stato di ogni singola istanza quando l'istanza viene creata, ad es. nello spazio nomi `::tpad::.edit1` per l'istanza .edit1.

Nello stesso modo sono memorizzate e copiate le direttive di configurazione relative a ciascun plugin, a partire dal namespace che rappresenta il plugin, ad es. `::tpad::GNUfmt`.

Dalle considerazioni fatte, si capisce che la API offerta da tpad, basata principalmente sui namespace dinamici di Tcl, ha una struttura orientata agli oggetti, pur non essendo basata su nessuna estensione object-oriented di Tcl. Le istanze di tpad sono simili agli oggetti e il pathname Tk che li identifica è come il nome unico di una variabile oggetto. I namespace dinamici di Tcl sono usati per contenere diversi oggetti e gli operatori (comandi) che agiscono su di essi, ed essendo degli insiemi di oggetti simili con operazioni, corrispondono al concetto di classe. Il comando di inizializzazione che crea l'istanza è l'analogo di un costruttore. Gli altri comandi pubblicamente esportati dal namespace `::tpad` si possono definire metodi pubblici e quelli non esportati, metodi privati. I comandi esportati che non hanno un parametro che identifica l'istanza di tpad su cui agire sono come i metodi statici di una classe. Il pathname Tk che identifica l'istanza di tpad su cui operare equivale alla variabile `this` che si riferisce all'oggetto corrente. Le variabili o costanti di configurazione relative a tutte le istanze di tpad o di un suo plugin sono l'analogo delle variabili o costanti statiche di una classe.

1.6 Controllo di tpad dall'interno di altre applicazioni

Un notevole beneficio si ha quando un'applicazione, sia scritta in Tcl puro sia in qualche altro linguaggio di sistema che può incorporare Tcl, implementa, oltre che la sua GUI, anche le sue principali funzionalità tramite estensione del linguaggio Tcl, ovvero tramite una API Tcl. In tal caso il linguaggio Tcl stesso potrà essere usato anche per il controllo dell'applicazione da parte di un'altro processo che gira sullo stesso display.

Questo avviene tramite il comando `send`, che riceve come argomenti un nome che identifica univocamente un'applicazione e un comando Tcl qualsiasi, con i suoi argomenti, da eseguire nell'applicazione identificata. Si tratta di una forma di comunicazione simile alle chiamate di procedura remote, ma limitata ad applicazioni che hanno la finestra grafica principale sullo stesso display.

L'invocazione può essere fatta in modo *sincrono* o *asincrono*. Nel modo sincrono, che è quello di default, l'applicazione che manda il comando ad un'altra si mette in attesa che il comando inviato sia stato eseguito. In tal caso `send` ritornerà il risultato del comando (od eventuali informazioni di errore) all'applicazione invocante. Nel modo asincrono invece il comando `send` ritorna immediatamente senza alcuna attesa da parte del chiamante e le informazioni di ritorno dal comando chiamato vengono scartate.

Gli usi del comando `send` sono innumerevoli. Ad esempio i demo Tk includono un'applicazione detta `tcolor` che è un semplice color editor. Nella [Figura 1.9] vediano un esempio di come `send` possa essere usato per la riconfigurazione a run-time delle GUI definite tramite Tcl: l'utente avvia tcolor, sceglie un nuovo colore e poi manda ad un'altra applicazione, nell'esempio a tpad, un semplice comando Tcl per impostare lo sfondo del testo con il colore scelto.

La [Figura 1.10] mostra un altro esempio di comunicazione tra applicazioni.

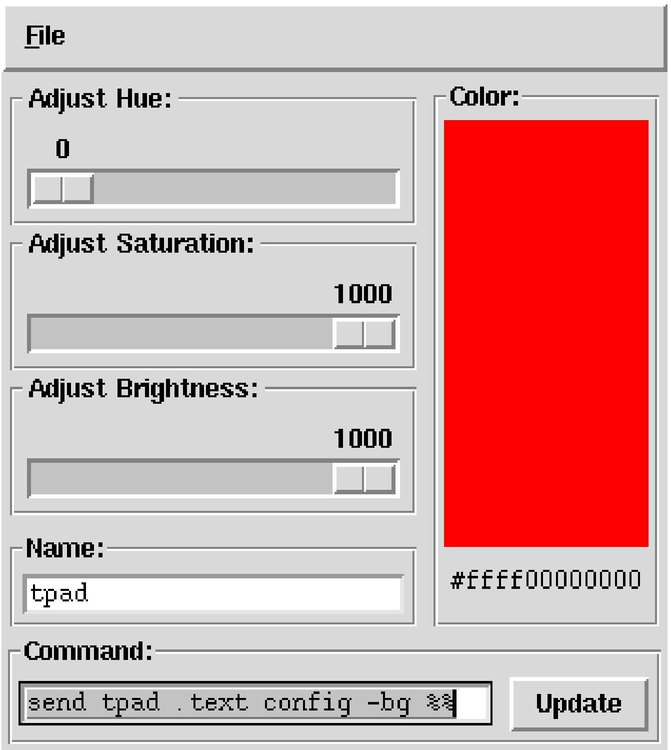

Figura 1.9: L'applicazione tcolor può essere usata per cambiare i colori di un'altra

Per la simulazione ho utilizzato `rmt`, una semplice applicazione demo di Tk che consente, tramite una comoda GUI, di scegliere un'applicazione destinataria e inviarli dei comandi. I comandi visibili nella [Figura 1.10], mostrano come un ipotetico debugger possa comunicare con tview, la versione di tpad che opera come solo visualizzatore di testo. Il debugger dapprima comanda a tview di aprire il file sorgente e poi gli chiede di evidenziare la linea corrente di codice che dovrà essere eseguita durante un'esecuzione passo passo.

1.7 Utilizzo di tpad in un'architettura client-server grafica

Il sistema grafico network-trasparente X Window, chiamato anche X11 o semplicemente X, disponibile per un'ampia gamma di sistemi operativi e architetture hardware, rende possibile l'utilizzo delle sue applicazioni grafiche attraverso la rete. Questo vale anche per le applicazioni Tcl/Tk in Unix [§ 2.4.3, p. 2.4.3].

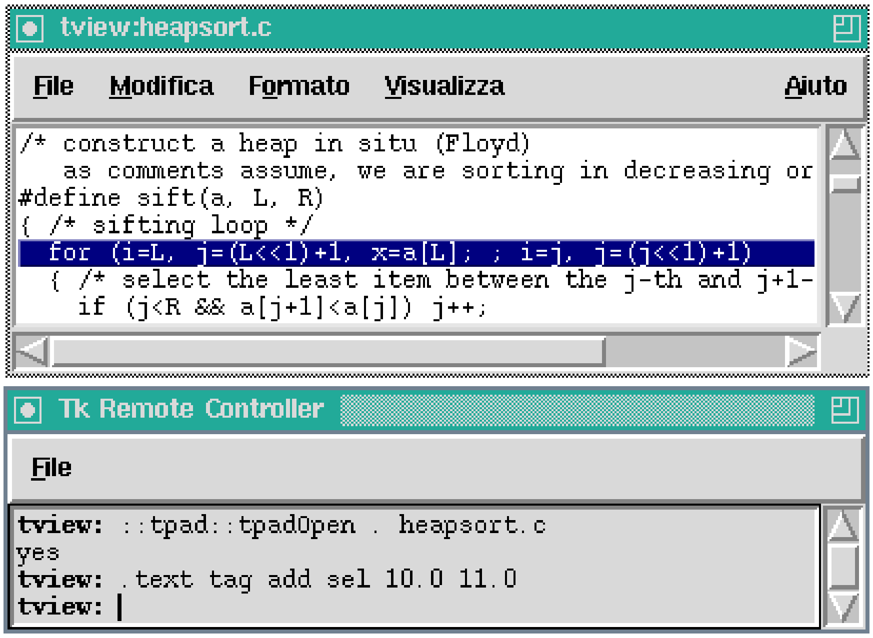

Figura 1.10: Simulazione dell'uso di tpad da parte di un debugger

Con il termine *stazione grafica* o *display X* si intende l'insieme di una tastiera, uno o più monitor e un mouse utilizzati insieme per l'interazione. In genere una workstation ha un solo display (con un solo monitor) e quindi può essere usata da un solo utente alla volta, ma X supporta anche situazioni più complesse, con più di un display e con più di un monitor appartenente a ciascun singolo display grafico.

Il sistema X è suddiviso in vari pacchetti contenenti non solo programmi eseguibili, ma anche font, icone, file di configurazione, librerie condivise con relativi file header e documentazione per utenti e programmatori.

Un servente X (X server) si occupa della gestione di un unico display e contiene i driver specifici per far funzionare l'hardware utilizzato. Nel caso un calcolatore abbia più di un display, si devono avviare più server X, uno per ciascun display che si vuole controllare.

Ciascun server X mette a disposizione le primitive per il disegno e gestione degli eventi tramite un opportuno protocollo, permettendo solo in questo modo l'accesso al display ai programmi applicativi.

Quest'ultimi sono detti clienti X (X client) e possono girare sullo stesso calcolatore a cui il display controllato dal server X è fisicamente connesso o su calcolatori differenti connessi tramite reti TCPIP o DECnet. I client X si connettono al server X locale o remoto e per gestire l'interazione con l'utente utilizzano le funzioni delle librerie client X (Xlib) che implementano il protocollo X nascondendone la complessità.

Persino il gestore di finestre (window manager) non è altro che un client X speciale e come tutti i client X è un programma separato dal server X. Infatti ne esistono svariati tipi intercambiabili. Tutti implementano comunque le funzionalità di base che consistono nel permettere all'utente di passare da un'applicazione X ad un'altra e di aprire, chiudere, ridimensionare, spostare e ridurre ad icona le finestre delle applicazioni.

Ho sperimentato l'utilizzo di tpad attraverso la rete, facendo uso di un client X Linux e una macchina Windows XP nel ruolo di server X.

In Linux al minimo occorre installare: tpad, l'interprete Tcl/Tk e le librerie di XFree86 [13], una versione open-source di X. Non è necessario installare un server X, a meno che non si voglia utilizzare una eventuale console grafica direttamente sul server.

In Windows occorre installare un server X e le relative librerie client Xlib, che sono necessarie anche per i server X. Ho scelto XWin, la port di XFree86 per questo sistema che si basa sul layer di compatibilità Unix Cygwin.

La macchina Windows va poi configurata per permettere l'accesso controllato del computer con Linux al suo display con uno dei metodi di autenticazione messi a disposizione da X Window, che variano a seconda del livello di sicurezza richiesto. Il metodo più semplice è descritto nella [§2.4.3, p. 44].

Dato che il computer Linux agiva anche da server web, questa configurazione è risultata molto comoda per editare i file HTML a livello di sorgente direttamente sul server web, senza necessità di continui trasferimenti tramite FTP. Tpad risulta adatto per questo scopo in quanto è molto simile a Notepad, quindi è già familiare a tutti gli utenti Windows, e inoltre è dotato di correttore HTML integrato [§ 1.4.2].

1.8 Sviluppi futuri

Un programma non è mai finito, ma
da qualche parte bisogna pur fermarsi.

— Peter Norton

Dal punto di vista dell'utente le prossime versioni vedranno implementate molte nuove funzionalità di editing e nuovi plugin. Il file "TODO" reperibile nel sito ufficiale [4] presenta una lista dettagliata degli sviluppi futuri.

Dal punto di vista del programmatore, la API di tpad [§ 3.1.2, p. 50] sarà riscritta utilizzando l'estensione object-oriented Itcl. La nuova API rimpiazzerà l'attuale sua mimica basata solo namespace dinamici e Itcl diventerà una dipendenza obbligatoria di tpad e di tutte le applicazioni che lo incorporeranno. La attuale API è un po' inconsistente dal punto di vista sintattico con il resto del linguaggio soprattutto perchè non ha una sintassi di comandi con parametri con nome, bensì anonimi. Ho in progetto di uniformare completamente la sintassi con quella degli altri comandi standard di Tcl.

Attualmente la libreria tpad permette la creazione di editor capaci di editare solo testo semplice. In futuro verrà estesa in modo che possa essere usata da supporto per la creazione di sistemi di word-processing WYSIYWG.

Al contrario di quello che si potrebbe pensare a prima vista, questa evoluzione non richiederà molti sforzi e sarà in gran parte indolore: non sarà infatti necessario complicare la API di tpad, che rimarrà semplice ed elegante com'è attualmente, grazie al fatto che il widget `text` di Tk su cui tpad è basato e la libreria sono del tutto disaccoppiati. Inoltre il widget `text` già permette di visualizzare diverse porzioni di testo con differenti font e colori, di incorporare delle immagini nel testo e persino creare sistemi di navigazione ipertestuali.

La libreria pertanto dovrà subire solo poche modifiche interne per supportare queste nuove funzionalità. Per esempio le funzioni di lettura e salvataggio dei testi dovranno essere modificare per consentire la lettura e la scrittura di file con formato di testo formattato, ma la loro interfaccia rimarrà retrocompatibile.

Tutto questo vedrà la luce nella versione 2.0 di tpad.

Capitolo 2

Documentazione per l'utente

Nelle sezioni seguenti viene presentata una versione tradotta dall'inglese in italiano e leggermente adattata di quasi tutta la documentazione in linea di tpad, scritta per gli utenti non programmatori.

2.1 Configurazione

La configurazione di tpad avviene editando il file `tpad.conf`. Questo file utilizza la sintassi del linguaggio Tcl, pertanto occorre fare attenzione a non commettere errori di sintassi, altrimenti l'editor non si avvierà. Per contro il Tcl fornisce ampie possibilità di riconfigurazione. Si consiglia di creare una copia di backup del file `tpad.conf` prima di modificarlo, oppure di applicare le modifiche ad una sua copia, controllando che sia sintatticamente corretta (eseguendola tramite l'interprete Tcl) prima di sostituirla al vero file di configurazione. A tale scopo nei sistemi Unix si possono utilizzare comandi simili ai seguenti:

```
$ cp tpad.conf tpad.conf.default
$ cp tpad.conf tpad.conf.copy
... editing di tpad.conf.copy
$ tclsh8.4 tpad.conf.copy
$ mv tpad.conf.copy tpad.conf
```

Per operare cambiamenti alla configurazione che riguardano solo un singolo utente, piuttosto che modificare il file di configurazione globale, basta copiare quest'ultimo nella home directory dell'utente come file nascosto di nome e percorso `~/.tpadrc` e quindi modificare solamente questo file. Qualora qualcosa dovesse andare storto e tpad non si avvii più e non se ne riesca a capire il motivo, si può semplicemente cancellare il file `~/.tpadrc` e ricominciare daccapo.

All'avvio tpad ricerca per primo il file `~/.tpadrc`; solo se non riesce a trovarlo, tenta di caricare il file di configurazione globale `tpad.conf`, pertanto `~/.tpadrc`, se esiste, si sostituisce completamente a `tpad.conf`. Questo implica che tutte le opzioni di configurazione presenti nel file globale sono richieste anche in quello per utente. In altri termini nessuna delle opzioni dovrebbe essere commentata o cancellata, altrimenti tpad non si avvierà.

Segue il file di configurazione di default incluso nella versione corrente di tpad:

```
# tpad.conf --
#
#   This is the global default configuration file for tpad(1).
#   You can safely change constants and default values
#   of variables below as long as you make no syntax errors.
#   See Tcl(n) for a summary of Tcl language syntax and
#   tpad documentation for more information on the
#   configuration options below.
#

## Initialize tpad static vars or constants.

variable version 1.3  ;# don't change this

# file types listbox entries
variable types {
  {{All Files}      *                         }
  {{Text Files}     {.txt .text}              }
  {{HTML Files}     {.html .htm .shtml .xhtml}}
  {{XML Files}      .xml                      }
  {{TCL Files}      {.tcl .tk}                }
  {{C Source Files} {.c .h}                   }
}

# include tearoff in pulldown menus (no, yes)
variable tearoff no

variable palette gray85  ;# color scheme (see tk_setPalette(n))
variable fg black        ;# text and entry foreground color
variable bg white        ;# text and entry background color
variable selfg white     ;# selection foreground color
variable selbg navy      ;# selection background color

# font to use for the editor (a fixed size font is recommended)
```

```
variable editfont Courier
variable statusfont Helvetica  ;# font to use for the status bar

# lpr(1) like command and options to use for off line printing
variable lpr {lpr}

variable logfac yes  ;# turn on log creation facility (no, yes)

variable timefmt %c  ;# date and time format
variable titlefmt {%1$s:%2$s}  ;# title bar format

variable insertofftime 300  ;# cursor off time (milliseconds)
variable insertontime 600   ;# cursor on time (milliseconds)

# width/height of vertical/horizontal scrollbars (distance)
variable scrollwidth 15

# Warning: there seems to be a bug with the undo implementation
# in Tk 8.4.6 or previous, which causes segmentation faults when
# this parameter is set to more than one.
# This is not tpad's fault and I hope will be fixed soon.
# To be safe, only use the values 1 and 0 (unlimited).
variable maxundo 1  ;# maximum number of compound undo actions

# command to run an external browser to read the HTML help
variable browsercmd mozilla

# path to where the HTML documentation lives
variable helppath {/home/ant/tpad-all/share/doc/tpad}

# initial placing of non-modal dialogs ({},pointer,widget,none)
variable placing widget

# max entries of the "open recent" list (0 for no list)
variable maxopenrec 10

# file where the "open recent" list is saved
# ({} for non-permanent list)
variable openrecfile {~/.tpad_history}

# xfontsel(1) path (empty if you don't have it)
variable xfontsel {xfontsel}
```

```
variable fmtkind BSD  ;# kind of fmt(1) (GNU, BSD, none)
variable fmt {fmt}    ;# fmt(1) path (ignored if fmtkind is none)

# xmlwf(1) path (empty if you don't have it)
variable xmlwf {xmlwf}

# note that you will also need to install Tcllib
# for this plugin to work
variable tidy {tidy}  ;# tidy(1) path (empty if you don't have it)

# display fortune tip at startup (no, yes)
variable tip yes

# you will need to change this only if you move or
# rename the fortune database file and its index
# (which has the added extension .dat)
variable fortunefile {/home/ant/tpad-all/share/tpad/tips/tpad-tips}

set tk_strictMotif 0  ;# strict Motif compliance (0, 1)

## End of static vars or constants initialization.


## Initialize tpad instance vars to default values.

variable tool yes    ;# display tool bar (no, yes)
variable wrap none   ;# wrap mode (none, word)
variable pattern {}  ;# search pattern
variable substr {}   ;# substitution string
variable dir down    ;# search direction (up, down)
variable case no     ;# case sensitive search (no, yes)
variable regexp no   ;# match using Regular Expressions (no, yes)
variable status yes  ;# display status bar (no, yes)

variable ascii no   ;# display ascii table (no, yes)
variable format no  ;# display text formatter window (no, yes)
variable wf no      ;# display xmlwf window (no, yes)
variable ht no      ;# display HTML Tidy window (no, yes)

# default encoding to use when reading files
variable inputEnc system
# default encoding to use when writing files
variable outputEnc system
```

```
## End of tpad instance vars initialization.


## Initialize ascii plugin constants.

namespace eval ascii {
  variable font Courier  ;# font to use for table characters
}

## End of ascii plugin constants initialization.


## Initialize BSD fmt(1) plugin vars to default values.

namespace eval BSDfmt {
  variable width 65      ;# line width alias goal length
  # excess chars so that maximum=width+excess
  variable excess 10
  variable center no     ;# center the text (no, yes)
  # don't ignore nroff command lines (no, yes)
  variable nonroff no
  variable mail no       ;# try to format mail headers (no, yes)
  # collapse whitespaces inside lines (no, yes)
  variable spaces no
  variable ending {.?!}  ;# sentence ending chars
  # replace spaces with tabs at the start (no, yes)
  variable replace no
  variable para no       ;# allow indented paragraphs (no, yes)
}

## End of BSD fmt(1) plugin vars initialization.


## Initialize GNU fmt(1) plugin vars to default values.

namespace eval GNUfmt {
  variable width 75    ;# maximum line length
  variable uniform no  ;# uniform spacing (no, yes)
  # format only lines beginning with this prefix
  variable prefix {}
  variable crown no    ;# crown margin mode (no, yes)
  variable tagged no   ;# tagged paragraph (no, yes)
```

```
  # split long lines, but do not refill (no, yes)
  variable split no
}

## End of GNU fmt(1) plugin vars initialization.


## Initialize xmlwf(1) plugin vars or constants to default values.

namespace eval xmlwf {
  variable font Helvetica  ;# font to use for error messages

  variable external no    ;# parse external entities (no, yes)
  variable parameter no   ;# process parameter entities (no, yes)
  variable standalone no  ;# error if not standalone (no, yes)
  # process (describe) namespaces (no, yes)
  variable namespace no
}

## End of xmlwf(1) plugin vars or constants initialization.


## Initialize tidy(1) plugin vars or constants to default values.

namespace eval tidy {
  variable errorFg white
  variable errorBg #b03060
  variable warningFg white
  variable warningBg #303080
  # used for the show/hide options button
  variable smallFont {Helvetica -9}

  # whether or not display options at startup (no, yes)
  variable showopts yes

  # additional config file: if not empty this value overrides
  # the environment variable HTML_TIDY; if empty and HTML_TIDY
  # is defined, the value of HTML_TIDY will be assigned to it;
  # please always use an absolute path
  variable config {}     ;# {} if none or default

  variable indent no     ;# indent contents of elements (no, yes)
  variable wrap yes      ;# wrap output (no, yes)
```

```
  # default right margin for line wrapping
  variable column 68
  # preserve source file entities as is (no, yes)
  variable preserve no
  variable xml no        ;# input is wellformed xml (no, yes)
  variable omit no       ;# omit optional endtags (no, yes)
  variable upper no      ;# force tags to upper case (no, yes)
  # replace font, nobr & center tags by CSS (no, yes)
  variable clean no
  # convert html to wellformed xml (no, yes)
  variable asxml no
  # output numeric rather than named entities (no, yes)
  variable numeric no
}

## End of tidy(1) plugin vars or constants initialization.
```

La modifica del file di configurazione non richiede una conoscenza approfondita di Tcl, dal momento che utilizza solo pochi comandi del linguaggio.

È sufficiente notare che in Tcl i valori booleani si possono indicare sia come `0` e `1` sia come `yes` e `no` e che le stringhe composte da più di una parola vanno racchiuse tra parentesi graffe (ad es. `{word1 word2}`) in modo che Tcl le riconosca come un unico argomento di comando.

La lista seguente descrive in dettaglio il significato di ciascuna costante o variabile, indicando i possibili valori che può assumere tra parentesi graffe. I comandi Tcl a cui si fa riferimento sono sempre seguiti dall'indicazione della sezione del manuale Unix in cui si trovano tra parentesi tonde che è quella riservata ai nuovi comandi. Ad esempio: `variable(n)` (dove n sta appunto per *new*).

version {numero di versione} Contiene il numero di versione di tpad. Non dovrebbe essere cambiata dagli utenti, ma solo dagli sviluppatori.

types {liste innestate di Tcl} Definisce la lista dei tipi di file da aprire nella finestra di dialogo "Apri". Le liste innestate sono definite come in Lisp, eccetto che come caratteri di quotatura innestabili si usano le parentesi graffe anziché le tonde. Per maggiori dettagli, vedere la pagine di manuale del comando `tk_getOpenFile(n)` o `tk_getSaveFile(n)`, in particolare la sezione intitolata SPECIFYING FILE PATTERNS.

tearoff {no,yes} Specifica se i menu sono del tipo staccabile, cioè se è possibile creare delle finestre a sé stanti con una copia di un menu e di tutti i suoi sottomenu. Staccare un menu risulta utile se qualcuna delle sue opzioni si utilizza frequentemente. Persino il menu di popup può essere staccato.

palette {specifica di palette} Definisce lo schema dei colori dell'applicazione. Tk ha molti schemi predefiniti che si possono selezionare semplicemente indicando il nome di un colore di base da usare come colore di sfondo normale (vedere `colors(n)`); la palette dei colori verrà dunque calcolata a partire da questo colore di base. Se invece si preferisce definire i colori uno per uno, vedere `tk_setPalette(n)` per la sintassi. Notare che le opzioni **fg**, **bg**, **selfg** e **selbg** che seguono ridefiniscono alcuni dei colori dello schema scelto tramite questa opzione e, come per tutte le opzioni, è obbligatorio specificarne un valore.

fg {nome di colore} Colore di primo piano del testo. Vedere `colors(n)` per i nomi dei colori. Per default il testo è nero su sfondo bianco. Se si cambiano i colori, assicurarsi che il contrasto sia sufficiente, altrimenti il testo potrebbe essere invisibile o quasi.

bg {nome di colore} Colore di sfondo del testo. Idem.

selfg {nome di colore} Colore di primo piano della selezione. Idem.

selbg {nome di colore} Colore di sfondo della selezione. Idem.

editfont {descrizione di font} Tipo di carattere dell'area del testo e sua dimensione. Prima di cambiare il valore si raccomanda di leggere la pagina di manuale `font(n)`, per lo meno la sezione FONT DESCRIPTION. Per ottenere i migliori risultati e prestazioni, scegliere un font che sia in grado di visualizzare tutti i glifi della codifica di I/O prescelta.

statusfont {descrizione di font} Font da usare per la barra di stato. Idem.

lpr {comando di shell} Questo è il comando che viene invocato dall'opzione "Stampa" nel menu "File" per inviare il testo contenuto nel buffer interno di tpad alla stampante. Il comando deve essere di tipo filtro, come `lpr(1)` di Unix. Vedere la manpage di `lpr` per la descrizione delle varie opzioni che si possono aggiungere. Per esempio in FreeBSD si può utilizzare il filtro `pr(1)` per paginare l'output semplicemente aggiungendo ad `lpr` l'opzione `-p`, ottenendo il comando `{lpr -p}`. Questa variabile può essere cambiata anche a runtime utilizzando l'opzione del menu "File/Imposta stampante".

logfac {no,yes} Abilita/disabilita la funzionalità di creazione dei log [§ 2.6, p. 48]. Per default è abilitata, in quanto risulta utile per scrivere dei file di log o diari. Di solito si desidererà disabilitarla se tpad non viene usato come editor generico ma incorporato in qualche altra applicazione in cui non si vuole che sia attiva.

timefmt {formato del tempo} Una stringa che descrive il formato della data e dell'ora inseriti tramite la funzione "Modifica/Ora/Data". Fare riferimento alla pagina di manuale `clock(n)` (vedere l'help del sottocomando `clock` `format`). Il valore di default è `%c` che indica un formato dipendente dalle impostazioni della lingua corrente (*locale*) del sistema.

titlefmt {stringa di formato} Controlla il formato della barra del titolo, utilizzando una stringa di formato nello stile del comando `format(n)`. Il segnaposto `%1$s`, se presente, verrà sostituito dal nome dell'eseguibile (*tpad* oppure *tview*), mentre `%2$s` viene sostituito dal nome del file che si sta editando.

insertofftime {millisecondi} Specifica per quanti millisecondi (valore intero non negativo) il cursore di inserzione deve essere spento in ciascun ciclo di lampeggiamento. Settare questa opzione a zero per far sì che il cursore non lampeggi. Nel caso di utilizzo di tpad in remoto, specie da parte di molti utenti, questo può essere utile per evitare di avere un flusso costante di traffico di rete anche quando un utente non è in attività, dal momento che la linea del display su cui si trova il cursore viene ridisegnata ad ogni lampeggiamento.

insertontime {millisecondi} Specifica per quanti millisecondi (valore intero non negativo) il cursore di inserzione deve rimanere accesso in ciascun ciclo di lampeggiamento.

maxundo {numero intero} Indica il massimo numero di azioni composte sullo stack di undo. Un valore zero o negativo implica uno stack di annullamento potenzialmente illimitato. Naturalmente più grande diventa lo stack di undo, maggiore sarà la memoria usata dal programma. Per default questo valore è 1, che significa che si può annullare o ripetere solo l'ultima azione di editing.

browsercmd {linea di comando} Questo comando viene utilizzato per lanciare un browser esterno quando viene selezionata l'opzione "Aiuto/Aiuto" del menu. Si possono anche specificare eventuali opzioni della riga di comando previste dal browser scelto.

helppath {percorso di directory} Indica il percorso della directory in cui si trovano i file del manuale di tpad in formato HTML. A questo percorso viene concatenato il nome del file contenente la pagina indice (`index.html`) e il risultato viene passato come ultimo argomento al comando specificato dalla costante **browsercmd** di cui sopra.

placing {{},pointer,widget,none} Stabilisce come devono essere posizionate le finestre figlie di tpad sullo schermo la prima volta che vengono visualizzate. I valori possibili sono dati dalla tabella seguente:

valore	significato
{}	posiziona le finestre figlie al centro dello schermo
`pointer`	posiziona le finestre figlie in modo che la posizione corrente del puntatore si trovi al centro della finestra
widget	posiziona le finestre figlie centrandole rispetto alla finestra principale di tpad
none	non imporre nessun posizionamento, lascia la scelta al gestore delle finestre

L'opzione `none` è utile solamente quando si usa un window manager abbastanza intelligente. Notare che questa impostazione si applica solo alle finestre di dialogo non modali, mentre quasi tutte le finestre di dialogo modali vengono sempre centrate rispetto alla finestra principale di tpad, qualsiasi sia il valore di questo parametro.

maxopenrec {numero intero} Massimo numero di elementi della lista "Apri recenti"; il valore di default è 10, che dovrebbe essere sufficiente per un utilizzo normale; il valore 0 indica di non tenere alcuna lista.

openrecfile {file path} Nome del file in cui salvare il contenuto della lista "Apri recenti"; per default è il file dell'utente `~/.tpad_history`, ma potrebbe anche essere un file globale purché ciascun utente abbia i diritti di scrittura su questo file; se questa opzione è la stringa vuota, la lista sarà di tipo non permanente; l'opzione viene ignorata se maxopenrc è 0.

fmtkind {GNU,BSD,none} `fmt(1)` è un semplice formattatore di testo che può essere utilizzato all'interno di tpad tramite un plugin [§ 1.4.4, p. 11]. Di solito, per default, Linux ha la variante GNU di `fmt`, mentre FreeBSD usa una versione differente. Assicurarsi di specificare correttamente il tipo di `fmt` che si ha, altrimenti non potrà essere invocato correttamente dall'interno di tpad. Se tpad è stato installato tramite un pacchetto specifico della piattaforma che si sta usando, il valore di default dovrebbe essere quello giusto.

fmt {percorso del comando} Dal momento che in Linux e FreeBSD `fmt(1)` è parte delle utilità standard di sistema, non è normalmente necessario specificare il percorso in cui si trova. Tuttavia questo può essere utile se si vuole usare un tipo di `fmt` differente che altrimenti non potrebbe essere invocato semplicemente per nome utilizzando il valore corrente del PATH o perchè ha un nome differente. Per esempio in FreeBSD, installando le core utilities della Free Software Foundation (`ports/sysutils/coreutils`), tra le altre cose, verrà installata la versione GNU di `fmt`, rinominata come `gfmt` per distinguerla da quella di BSD (che non viene rimossa). In tal caso, per usare l'`fmt` della GNU piuttosto che quello di default di FreeBSD, basta impostare questa variabile al valore `gfmt` e la variabile **fmtkind** al valore `GNU`.

tk_strictMotif {0,1} Va settata ad 1 se si vuole che tpad aderisca strettamente allo standard grafico Motif. Tenere presente che in modalità Motif non tutte le combinazioni di tasti funzionano. La manpage `Tk_StrictMotif(3)` contiene maggiori informazioni a proposito.

wrap {none,word} Indica come trattare le linee di testo che sono troppo lunghe per poter essere visualizzate in una singola linea della finestra del testo. Il valore `none` fa sì che ciascuna linea di testo appaia esattamente come una singola linea sullo schermo; i caratteri extra che non trovano posto sullo schermo non vengono proprio visualizzati. Nella modalità `word` invece ciascuna linea di testo verrà suddivisa in più linee sullo schermo se necessario, in modo che tutti i caratteri di cui è composta la linea siano visibili; la suddivisione verrà fatta solamente ai confini delle parole, cioè nessuna parola verrà mai spezzata su due linee.

pattern {stringa} Il pattern di ricerca di default. Di solito una stringa vuota.

substr {string} La stringa di sostituzione utilizzata dalla funzione "Sostituisci" del menu "Modifica". Per default è la stringa vuota.

dir {up,down} La direzione di ricerca di default. Quando una ricerca raggiunge l'inizio o la fine del file tpad emette un messaggio di avvertimento.

case {no,yes} Stabilisce se per default la ricerca sarà sensibile o meno alla differenza tra maiuscole e minuscole. Da impostare a seconda di quale tipo di ricerca si utilizza più di frequente in modo da minimizzare la necessità di cambiarla a runtime.

regexp {no, yes} Qualora utilizzate le espressioni regolari molto spesso, settate questa opzione a `yes` in modo che le funzioni di ricerca e sostituzione operino per default con espressioni regolari.

status {no,yes} Se visualizzare o meno la barra di stato per default.

ascii {no, yes} Impostare a `yes` per visualizzare il plugin tabella ASCII all'avvio.

format {no, yes} Impostare a `yes` per visualizzare il plugin formattatore di testo all'avvio.

wf {no, yes} Impostare a `yes` per visualizzare il plugin xmlwf di Expat all'avvio.

ht {no, yes} Impostare a `yes` per visualizzare il plugin di HTML Tidy all'avvio.

inputEnc {system,nome della codifica} Codifica di default da usare quando si legge un file. Il valore speciale `system` indica la codifica di sistema corrente ed è quello di default. Occorre fare attenzione se si setta questa opzione a valori differenti di `system`; infatti se si usa una codifica di lettura che non corrisponde a quella utilizzata dal documento che si sta editando, si possono perdere alcuni caratteri quando si legge o si salva un file (che verranno visualizzati o scritti su disco come un ?). Se qualche volta dovete modificare o visualizzare dei file che usano una codifica differente rispetto a quella utilizzata dal sistema, potete cambiare la scelta della codifica a runtime utilizzando l'opzione del menu "Formato/Codifica di I/O"; quando avete finito, utilizzate la stessa opzione, oppure uscite e rientrate in tpad per tornare alla codifica di sistema.

outputEnc {system,nome della codifica} Codifica di default da utilizzare per la scrittura dei file. Idem.

Per quanto riguarda la descrizione delle opzioni relative ai plugin, che sono raggruppate ciascuno in uno spazio nomi unico etichettato con lo stesso nome del plugin, si rimanda alla documentazione del tool su cui il plugin stesso fornisce l'interfaccia. Ad esempio, per consultare il manuale del text formatter, utilizzare uno dei seguenti comandi, a seconda della versione di Unix che si sta usando:

```
$ fmt -h
$ man fmt
$ info fmt
```

2.1.1 Aggiornamento della configurazione

Un problema comune che si verifica dopo aver aggiornato tpad all'ultima versione disponibile (così come pure per qualsiasi altro programma che sia configurabile) è che i vecchi file di configurazione solitamente vengono mantenuti e perciò devono essere aggiornati manualmente. Tpad non sarà in grado di partire se si verifica una delle seguenti condizioni, in ordine di gravità decrescente:

1. non riesce a trovare un file di configurazione

2. ha trovato il file di configurazione, ma si è verificato un errore nel momento in cui questo viene interpretato

3. ha trovato il file di configurazione, ma la versione del programma registrata nella variabile **version** non coincide con la versione del programma

Il caso 3 è appunto quello che si incontra comunemente dopo che è stato effettuato l'aggiornamento di tpad. Per risolvere il problema occorre effettuare un aggiornamento della configurazione.

Dal momento che tpad è attivamente sviluppato e il file di configurazione potrebbe variare da una versione all'altra, si raccomanda la seguente procedura di aggiornamento, da applicarsi sia al file di configurazione globale che a quelli per utente:

1. se il vecchio file di configurazione era stato cambiato rispetto alla versione di default, farne un backup, in modo da non perdere traccia delle modifiche apportate. In alternativa al backup si può anche prendere nota dei cambiamenti fatti.

2. copiare la nuova versione di default del file di configurazione sopra la vecchia, sovrascrivendola. Gli amministratori di sistema devono assicurare che il file di configurazione globale possa essere letto da tutti gli utenti interessati ad usare tpad con la configurazione di default del sistema.

3. modificare la nuova versione del file di configurazione copiata al passo 2 riproducendo i cambiamenti che erano stati fatti al vecchio file di configurazione.

2.2 Combinazioni di tasti

Note:

- I nomi dei tasti indicati in questa sezione sono quelli della tastiera americana standard. La corrispondenza con i tasti della tastiera italiana, quando questi esistono, è comunque abbastanza ovvia e nota a tutti gli utenti.

- Tutte le azioni che possono essere effettuate con l'uso del mouse possono essere svolte anche con l'uso della sola tastiera. Questo è utile in situazioni di emergenza in cui il mouse non dovesse funzionare. Alcune funzioni sono più comode da effettuarsi con il mouse, altre viceversa lo sono con la tastiera e altre ancora si possono effettuare solo con la tastiera.

- Il window manager (abbr. *wm*) potrebbe definire dei tasti scorciatoia che sono in conflitto con alcuni dei tasti definiti da tpad. Per esempio se il wm è configurato per utilizzare i tasti `Ctrl-Left` e `Ctrl-Right` per cambiare il desktop virtuale, questi tasti non potranno essere utilizzati per spostare il cursore di parola in parola in tpad. Questo problema putroppo non ha una soluzione semplice: se si vogliono usare entrambe le funzioni associate allo stesso tasto (quella di tpad e quella del wm) occorre riconfigurare i tasti o in tpad oppure nel wm, assegnandole a combinazioni differenti.

- Alcuni tasti o modificatori che sono legati a particolari azioni potrebbero non essere presenti sul tipo di tastiera che utilizzate. In questi casi c'è di solito un tasto o una combinazione di tasti equivalente. Per esempio se non c'è alcun tasto `Select`, se ne può ottenere lo stesso effetto con `Control-Space`.

2.2.1 relative alla selezione primaria

per incollare la selezione primaria

Il tasto `Insert` inserisce il contenuto della selezione primaria alla posizione corrente del cursore.

per impostare l'àncora di selezione

I tasti `Select` e `Control-Space` impostano l'àncora di selezione alla posizione corrente del cursore. La selezione corrente non viene alterata.

per aggiustare la selezione

I tasti `Shift-Select` e `Control-Shift-Space` aggiustano la selezione fino alla posizione corrente del cursore di inserzione, selezionando tutti i caratteri compresi tra l'àncora e il cursore di inserzione se precedentemente non vi era alcuna selezione.

per selezionare tutto il testo

Il tasto `Control-5` (del tastierino numerico) oppure `Control-/` seleziona l'intero contenuto del widget di testo.

per annullare la selezione

`Control-\` annulla qualsiasi selezione nel widget.

per cancellare la selezione

Il tasto `Delete` cancella la selezione, se ce n'è una nel widget. Se non c'è alcuna selezione, cancella il carattere alla destra del cursore di inserzione (come fa `Control-d`). Anche il tasto `Backspace` elimina la selezione, se questa è attiva. Se non c'è alcuna selezione, cancella il carattere alla sinistra del cursore di inserzione.

2.2.2 relative agli appunti

per copiare la selezione negli appunti

Il tasto `F16` (etichettato come `Copy` su molte workstation della Sun) oppure `Meta-w` o `Control-c` copia la selezione del widget negli *appunti* (*clipboard*), se questa è attiva.

per tagliare la selezione negli appunti

Il tasto `F20` (etichettato come `Cut` su molte workstation della Sun) oppure `Control-w` o `Control-x` copia la selezione del widget negli appunti e poi la cancella. Se non c'è alcuna selezione nel widget allora questi tasti non hanno alcun effetto.

per incollare la selezione negli appunti

Il tasto `F18` (etichettato come `Paste` su molte workstation della Sun) oppure `Control-y` o `Control-v` inserisce il contenuto degli appunti alla posizione corrente del cursore di inserzione.

2.2.3 per muovere il cursore di inserzione

di un carattere alla volta

I tasti `Left` e `Right` spostano il cursore di inserzione di un carattere a sinistra o a destra; inoltre annullano qualsiasi selezione sia attiva nel testo. Il tasto `Control-b` si comporta allo stesso modo di `Left`, mentre `Control-f` non va più a destra come nelle precedenti versioni, in quanto adesso serve per richiamare la finestra di dialogo "Trova". Se `Left` o `Right` vengono digitati quando è tenuto premuto il tasto `Shift`, allora quando il cursore di inserzione si muove la selezione viene estesa per includere anche il nuovo carattere.

di una parola alla volta

`Control-Left` e `Control-Right` muovono il cursore di inserzione di una parola alla volta e `Control-Shift-Left` e `Control-Shift-Right` muovono il cursore e inoltre estendono la selezione. `Meta-b` e `Meta-f` si comportano allo stesso modo di `Control-Left` e `Control-Right` rispettivamente.

di una linea alla volta

I tasti `Up` e `Down` spostano il cursore di inserzione di una linea sopra o sotto e annullano qualsiasi selezione attiva nel testo. Se `Up` o `Down` vengono digitati mentre è tenuto premuto il tasto `Shift`, allora quando il cursore di inserzione si muove la selezione viene estesa per includere anche i nuovi caratteri. `Control-p` si comporta allo stesso modo di `Up`.

di una paragrafo alla volta

`Control-Up` e `Control-Down` spostano il cursore di inserzione di paragrafo in paragrafo (un paragrafo è un gruppo di linee separate da una linea vuota), mentre `Control-Shift-Up` e `Control-Shift-Down` oltre a spostare il cursore estendono anche la selezione.

di una schermata alla volta

I tasti `Next` e `Prior` spostano il cursore di inserzione avanti e indietro di una schermata e annullano la selezione. Se usati in combinazione con il tasto `Shift`, la selezione viene estesa per includere i nuovi caratteri.

all'inizio della linea

I tasti `Home` e `Control-a` spostano il cursore di inserzione all'inizio della sua linea e annullano la selezione nel widget. `Shift-Home` muove il cursore di inserzione all'inizio della linea e inoltre estende la selezione fino a quel punto.

alla fine della linea

`End` e `Control-e` spostano il cursore di inserzione alla fine della linea e annullano la selezione nel widget. `Shift-End` muove il cursore fino alla fine della riga ed estende la selezione fino a quel punto.

all'inizio del testo

`Control-Home` e `Meta-<` spostano il cursore di inserzione all'inizio del testo e annullano la selezione nel widget. `Control-Shift-Home` sposta il cursore di inserzione all'inizio e inoltre estende la selezione fino a quel punto.

alla fine del testo

`Control-End` e `Meta->` spostano il cursore di inserzione fino alla fine del testo e annullano la selezione nel widget. `Control-Shift-End` sposta il cursore fino alla fine del testo ed estende la selezione fino a quel punto.

2.2.4 varie

per lo scorrimento laterale di una pagina

`Control-Next` e `Control-Prior` fanno scorrere la vista a destra o a sinistra di una pagina senza spostare il cursore di inserzione o modificare la selezione.

per cancellare fino alla fine della linea

`Control-k` cancella a partire dal cursore di inserzione fino alla fine della linea corrente; se il cursore si trova già alla fine della riga allora viene cancellato proprio il carattere di nuova riga.

per invertire l'ordine di due caratteri

`Control-t` inverte l'ordine dei due caratteri alla destra e alla sinistra del cursore di inserzione o dei due che si trovano alla sinistra se il cursore stesso si trova alla fine della linea. Questa funzione è utile per correggere rapidamente gli errori di battitura.

per annullare/ripetere un'azione di editing

se maxundo = 1 `Control-z` (e `Control-_` in Unix se l'opzione tk_strictMotif è impostata a `yes`) alterna l'esecuzione della funzione di annullamento e di ripetizione dell'ultima azione di editing. Non fa niente se il testo non è stato modificato.

se maxundo != 1 `Control-z` (e `Control-_` in Unix se l'opzione tk_strictMotif è impostata a `yes`) annulla l'ultima azione di modifica del testo. Non fa niente se lo stack di annullamento è vuoto. `Control-r` o `Control-Z` (oppure `Control-y` in Windows) riapplicano l'ultima azione annullata. Non hanno alcun effetto se lo stack di ripetizione è vuoto.

per suddividere una linea in due

`Control-o` apre una nuova riga inserendo un carattere di newline davanti al cursore di inserzione, senza spostarlo.

2.3 Funzioni del mouse

2.3.1 relative alla selezione primaria

per tracciare una selezione arbitraria

Trascinando il mouse col tasto `1` premuto si traccia una selezione primaria tra la posizione iniziale del cursore di inserzione e il carattere sotto il puntatore del mouse.

per selezionare

una parola Un doppio click col tasto `1` del mouse seleziona la parola sotto il cursore del mouse e posiziona il cursore di inserzione alla fine della parola.

più parole Trascinando il mouse dopo un doppio click si traccia una selezione primaria formata da intere parole.

una linea Un triplo click con il tasto `1` del mouse seleziona la linea sotto il cursore del mouse e posiziona il cursore di inserzione alla fine della linea.

più linee Trascinando il mouse dopo un triplo click si traccia una selezione primaria formata da intere righe.

per aggiustare un estremo della selezione

L'estremo della selezione primaria può essere regolato trascinando il mouse col tasto `1` mentre si preme il tasto `Shift`; questo aggiusterà l'estremo della selezione primaria che si trova più vicino al cursore del mouse quando è stato premuto il tasto `1`. Se prima di trascinare si fa un doppio click, la selezione primaria verrà regolata in unità di parole intere; se si fa un triplo click verrà regolata in unità di linea.

per incollare la selezione primaria

Se si fa click col tasto `2` del mouse senza muoverlo, la selezione primaria viene copiata nel testo alla posizione del cursore del mouse. Anche il tasto `Insert` della tastiera inserisce la selezione primaria, ma alla posizione del cursore di inserzione.

2.3.2 per posizionare il cursore di inserzione

Cliccando col tasto `1` del mouse si posiziona il cursore di inserzione giusto prima del carattere che si trova sotto il cursore del mouse, si assegna il focus di input all'area di testo, e si annulla qualsiasi selezione primaria attiva. Se si clicca col tasto `1` del mouse mentre si tiene premuto il tasto `Control` il cursore di inserzione verrà riposizionato senza che cambi la selezione primaria.

2.3.3 per lo scorrimento

senza selezione

La vista nel widget può essere regolata usando le scrollbar oppure trascinando il mouse col tasto `2` premuto.

con selezione

Se il mouse viene trascinato quando il puntatore si trova al di fuori dell'area di testo e col tasto `1` premuto, la casella di testo scorrerà automaticamente in modo da rendere visibile altre porzioni del testo (ammesso che ci sia del testo che si trova fuori schermo dal lato in cui il mouse ha lasciato la finestra).

2.4 Utilizzo dalla riga di comando

La riga di comando in Unix è tutt'altro che obsoleta. La shell è così flessibile e potente, soprattutto per automatizzare compiti ripetitivi che il più recente paradigma grafico di utilizzo del calcolatore basato sulle GUI non potrà mai eguagliarne tutte le funzionalità.

Per questa ragione si desidera che anche i programmi grafici in Unix possano essere controllati dalla riga di comando, per lo meno nella fase di avvio e tpad non fa eccezione a questa regola.

Per ottenere un sommario delle opzioni disponibili da riga di comando si usa l'opzione `-h`:

```
$ ./tpad -h
Application initialization failed: Command-specific options:
 -colormap: Colormap for main window
 -display:  Display to use
 -geometry: Initial geometry for window
 -name:     Name to use for application
 -sync:     Use synchronous mode for display server
 -visual:   Visual for main window
 -use:      Id of window in which to embed application
 --:        Pass all remaining arguments through to script

tpad Version 1.3
WinXP enhanced Notepad clone

usage: tpad [-Rh] [+#] [file ...]
```

Il modo più semplice di invocare tpad è di fornire solo il nome dell'eseguibile:

```
$ tpad
```

L'argomento addizionale sulla riga di comando usato più di frequente è il percorso del file da editare, che è opzionale (specificare il percorso è del tutto equivalente ad usare l'opzione "Apri" del menu "File" per caricare lo stesso file):

```
$ tpad myprog.c
```

Un'altra operazione comune che può essere effettuata tramite la linea di comando consiste nell'invocare l'editor e posizionare il cursore all'inizio di una certa linea. Si immagini, per esempio, che il compilatore C segnali che c'è un errore alla riga numero 23. Per correggerlo con tpad utilizzare questo comando:

```
$ tpad +23 myprog.c
```

Notare che i numeri di linea si contano a partire da 1 in quanto questa è la convenzione più comune. Per discriminare tra un numero di linea e il nome di un file si è scelto di utilizzare il segno più in quanto non è molto comune avere dei file che iniziano per `+`. Una volta avviato tpad si può sempre utilizzare l'opzione "Vai a" nel menu "Edit" per riposizionare il cursore all'inizio di una certa linea dato il suo numero.

Nei rari casi in cui si deve aprire un file il cui nome inizia con un segno `+`, utilizzare il percorso completo o quello relativo iniziante per `./` per evitare che il nome del file venga confuso con un numero di linea:

```
$ tpad ./+COMMENT
$ tview /var/db/pkg/tpad-1.2/+DESC
```

Se sapete in anticipo che avete da editare più di un file, potete decidere quali file modificare con un unico comando. Ecco un esempio che richiede l'editing di tre file chiamati `file1`, `file2` e `file3`:

```
$ tpad file1 file2 file3
```

Questo comando aprirà tpad caricando il file `file1` (se esiste). Terminato l'editing di `file1`, utilizzare l'opzione "Prossimo" nel menu "File" per caricare il prossimo file in sequenza (in questo caso `file2`) e così via.

Se si comanda di muovere il cursore ad una certa linea all'avvio e nello stesso tempo si richiede l'editing di file multipli in sequenza, l'opzione `+#` si applicherà solo al primo file della sequenza:

```
$ tpad +10 file1 file2 file3
```

e occorrerà utilizzare l'opzione "Vai a" del menu "Edit" per riposizionare velocemente il cursore nei file successivi.

Essendo uno script per la shell grafica `wish(1)`, tpad supporta anche tutte le opzioni della riga di comando descritte nel sommario OPTIONS della manpage di wish. Tutte queste opzioni possono essere mescolate insieme a quelle specifiche di tpad in qualsiasi ordine. Per esempio, supponiamo di voler editare il file `screen.txt` utilizzando una textarea di 80x25 caratteri (la stessa dimensione tipica di un terminale):

```
$ tpad -geometry 80x25 screen.txt
```

e poi di voler iniziare l'editing dalla linea numero 12:

```
$ tpad +12 -geometry 80x25 screen.txt
$ tpad -geometry 80x25 +12 screen.txt
```

2.4.1 Utilizzare tpad come visualizzatore

Quando tpad viene invocato come tview oppure se viene usata l'opzione `-R`, agisce da semplice paginatore o visualizzatore per uno o più file in sequenza. In questa modalità, non si può né inserire né modificare caratteri nel testo e non verrà visualizzato il cursore di inserzione. Le opzioni presentate dai menu saranno limitate a quelle che non apportano modifiche al testo (ad es. il comando "Salva" nel menu "File" non sarà disponibile).

Per esempio, si supponga di voler consultare il file di database degli utenti di Unix. In questo caso si può utilizzare l'uno o l'altro di questi comandi:

```
$ tpad -R /etc/passwd
$ tview /etc/passwd
```

Per evitare che un file possa essere modificato e salvato per errore, non è possibile passare dalla modalità di visualizzazione a quella di editing. Se proprio volete editare un file che state vedendo con tview, dovrete uscire da tview e caricare tpad senza l'opzione `-R` al suo posto.

Notare che tpad e tview non sono eseguibili separati e condividono lo stesso file di configurazione. Infatti tview è un link fisico o simbolico a tpad. Il tipo di link dipende dalla distribuzione di tpad utilizzata. Per maggiori informazioni sui link ai file in Unix consultare la pagina di manuale di `ln(1)`.

Naturalmente si possono anche vedere più file in sequenza, ad es:

```
$ tpad -R /etc/resolv.conf /etc/hosts
$ tview /etc/resolv.conf /etc/hosts
```

2.4.2 Invocare tpad da altri programmi

Un'altra questione connessa a come invocare tpad dalla linea di comando è come far sì che altri programmi invochino tpad. La maggior parte dei programmi che permettono ad un utente di editare un file utilizzando il suo editor preferito aderiscono alla seguente convenzione per stabilire quale editor esterno invocare: controllano la variabile di ambiente EDITOR; se questa non esiste, per default si invoca `vi(1)`, altrimenti si invoca il comando indicato in questa variabile. Per esempio, in Tcl questo comportamento si ottiene utilizzando il codice seguente:

```
set editor vi
if {[info exists env(EDITOR)]} {
  set editor $env(EDITOR)
}
```

Allora allo scopo di far utilizzare tpad al posto di `vi` o di qualcos'altro agli altri programmi, occorre assegnare il valore "tpad" alla variabile di ambiente EDITOR. Inoltre occorre che questa variabile venga marcata per l'esportazione automatica nell'ambiente dei comandi eseguiti successivamente (i cosiddetti *processi figli*). Questo si ottiene in modo diverso a seconda della shell che si sta utilizzando. Ecco come fare per i tipi di shell più comunemente usati:

sh, bash `export EDITOR=tpad`

csh, tcsh `setenv EDITOR tpad`

tclsh, wish `set env(EDITOR) tpad`

Infine, se si vuole che la variabile EDITOR venga impostata automaticamente ad ogni login, aggiungere l'istruzione che la imposta nel file di configurazione per utente della shell. Anche il modo per fare questo varia a seconda del tipo di shell:

sh `~/.profile`

bash `~/.bashrc`

csh, tcsh `~/.cshrc`

tclsh `~/.tclshrc`

wish `~/.wishrc`

2.4.3 Esecuzione remota

Per esemplificare supponiamo di lavorare su un computer chiamato `alpha` e di voler eseguire una copia di tpad installata su un altro computer detto `beta`, utilizzando il display di `alpha` per interagire col programma. Si suppone inoltre che entrambi i calcolatori siano connessi tramite una rete TCP/IP e che `alpha` e `beta` siano proprio i loro nomi host TCP/IP.

Con i termini *display* oppure *X display* si intende l'insieme di una tastiera, uno o più monitor e un mouse, utilizzati insieme per interagire con un calcolatore. Qualora `alpha` abbia più di un display, si può distinguere tra un display e l'altro grazie al fatto che sono numerati a partire da 0 (si indicano con `:0`, `:1`, `:2`, ecc). Se la tastiera e il mouse che appartengono al display numero `:X` controllano più di un monitor, si numerano i monitor a partire da 0 per distinguerli (e si scrive `:X.0`, `:X.1`, `:X.2`, ecc). La situazione più semplice e più comune è che `alpha` abbia un solo display (`:0`) con un singolo monitor (`:0.0`). In tal caso da `beta` ci si può riferire al display di `alpha` tramite la stringa `alpha:0` oppure `alpha:0.0`. Localmente da `alpha` si può anche omettere il nome host e riferirsi al display locale semplicemente con `:0` oppure `:0.0`.

Notare che anche se avete un solo display fisico, potete lo stesso avere più sessioni X virtuali che girano sullo stesso display, per esempio sotto utenti differenti e gestite da diversi window manager. La situazione è del tutto analoga ad avere più display fisici, eccetto che naturalmente solo un display virtuale alla volta può essere usato. Per esempio, per avviare una nuova sessione X sul display `:1` con lo script di sistema `startx(1)` basta fare:

```
(alpha) $ exec startx -- :1
```

Detto questo, la ricetta per avviare tpad (come qualsiasi altra applicazione X) in remoto è la seguente:

1. Su `alpha`, occorre autorizzare `beta` a stabilire connessioni al server X di `alpha`:

   ```
   (alpha) $ xhost +beta
   ```

2. Da `alpha`, effettuare il login a `beta`, utilizzando uno dei metodi disponibili, per esempio `telnet(1)`, `rlogin(1)` o `ssh(1)`.

3. A questo punto si possono scegliere due metodi alternativi per indicare a `beta` quale display remoto utilizzare per eseguire tpad (o qualunque altro programma X):

   - Impostate ed esportate la variabile d'ambiente `DISPLAY`, utilizzando l'apposito comando fornito dalla vostra shell (vedere la [§ 2.4.2] per maggiori informazioni su questo punto). Per esempio, supponendo di usare `sh` oppure `bash` e che `alpha` abbia un singolo display con un solo monitor:

     ```
     (beta) $ export DISPLAY=alpha:0
     (beta) $ tpad ...
     ```

   - Utilizzare l'opzione `-display` che ha precedenza sul valore della variabile di ambiente `DISPLAY`:

     ```
     (beta) $ tpad -display alpha:0 ...
     ```

Notare che in questo modo il computer `alpha` viene utilizzato solo come un display o terminale grafico remoto: è la CPU di `beta` che fa tutto il lavoro e carica e salva i file modificati, agendo direttamente sul filesystem di `beta` (a meno che `alpha` non esporti parti del suo filesystem verso `beta` tramite NFS).

Risoluzione dei problemi

Se l'esecuzione in remoto non funziona, per primo occorre controllare che il server X su `alpha` si ponga in attesa di connessioni TCP/IP entranti. Per esempio in FreeBSD a questo scopo si usa il comando `sockstat(1)`:

```
(alpha) $ sockstat -4l
```

Invece in Linux si usa `netstat(1)`, come segue:

```
(alpha) $ netstat --ip -l
```

Se su `alpha` state usando `startx(1)` per inizializzare la sessione X, provate ad utilizzare l'opzione `-listen_tcp` dello stesso, per abilitare l'uso da remoto del display X che si sta inizializzando. Infatti l'accesso potrebbe essere disabilitato per default per motivi di sicurezza (così è per esempio in FreeBSD). Se continua a non funzionare, occorrerà contattare il proprio amministratore di rete, per assicurarsi che non ci sia un firewall che impedisca a `beta` di connettersi ad `alpha` sulla porta numero $6000 + x$, dove x è il numero del display di `alpha` che state utilizzando.

Consigli per rendere sicura la connessione X

Nella [§ 2.4.3] avevamo limitato l'accesso al nostro server X che gira su `alpha` a tutti gli utenti di `beta`, utilizzando il comando `xhost(1)` in questo modo:

```
(alpha) $ xhost +beta
```

Questa configurazione è abbastanza sicura solamente se siete solo voi l'unico utente del sistema `beta` oppure se vi fidate abbastanza di tutti gli altri utenti di `beta`. Negli altri casi, sarà necessario un meccanismo di controllo di accesso più sofisticato, la cui descrizione esula dagli scopi di questo documento. La manpage `Xsecurity(7)` che viene distribuita con XFree86 descrive in dettaglio tutti i sistemi di controllo di accesso disponibili.

2.5 tpad & HTML Tidy

HTML TIDY [11] è un validatore, correttore e formattatore di file HTML sviluppato da Dave Ragget del W3C.

Tpad ha un plugin che rende molto facile invocare Tidy, senza dover uscire dall'editor o ricordare le opzioni della riga di comando [Figura 1.4, p. 11].

Esistono due modi operativi: "Valida" e "Valida e correggi".

"Valida" non modifica mai il file. Si limita a mostrare tutti i messaggi di errore o di avvertimento generati da Tidy durante l'analisi del documento all'interno di un controllo listbox intitolato "Ultimi Messaggi di Errore e Consigli". Facendo click su un elemento di questa lista tpad posizionerà il cursore al punto esatto in cui si è verificato il problema. Tuttavia se inserite o cancellate delle linee di testo, occorrerà ripetere la validazione prima di procedere con la correzione del prossimo errore, allo scopo di riallineare il log degli errori con il testo appena modificato.

"Valida e correggi" come "Valida" emette il log degli errori, ma in più modifica il testo originale sostituendolo con la versione corretta. La sostituzione viene effettuata come un'unica operazione composta e può essere annullata o ripetuta tramite l'opzione "Modifica/Annulla". In questo caso il log degli errori non può essere utilizzato per posizionare il cursore alla riga e alla colonna in cui si è verificato l'errore e perciò cliccando su un elemento della lista non si produce alcun effetto.

Se si vuole preservare la disposizione originale del testo, occorre utilizzare solamente la modalità "Valida", che corrisponde all'opzione da riga di comando `-e` di Tidy. Questo ha alcuni svantaggi, dovuti al modo di operare di Tidy piuttosto che a tpad: occorre correggere gli errori manualmente e nessuna delle opzioni di Tidy si può applicare in questa modalità, con l'eccezione di `-xml` e di alcune opzioni più avanzate che potete definire in un file di configurazione esterno. Questo perchè non viene generata alcuna versione formattata del documento: Tidy viene in sostanza utilizzato solo per fornire indicazioni su come correggere gli errori, che comunque risulta lo stesso molto utile.

Se invece preferite che sia Tidy ad effettuare tutto il lavoro di correzione e formattazione dell'HTML al posto vostro, utilizzate la modalità "Valida e correggi". A questa modalità si applicano tutte le opzioni del programma. L'unico svantaggio di questa modalità è che perderete il layout originale del sorgente. Ma questo inconveniente non è poi così grave come sembra, in quanto è più facile lasciare che sia Tidy ad occuparsi di applicare una consistente indentazione e a spezzare il testo sorgente sul margine destro, anziché farlo voi manualmente. Notare che se il documento ha degli errori che Tidy non può sapere come correggere (ad es. tag sconosciuti o valori di attributi non correttamente terminati), non sarà in grado di generare una versione riformattata del documento. In questi casi, la funzione "Valida e correggi" si comporterà esattamente come "Valida" e gli errori dovranno essere corretti manualmente sulla base delle indicazioni fornite da Tidy.

Installazione

Dal momento che Tidy non è una dipendenza di tpad, il plugin di tidy non è abilitato per default. Si prega di seguire la seguente semplice procedura per abilitarlo:

1. installare sia Tidy sia la libreria Tcllib (che è richiesta da questo plugin)

2. modificare il file di configurazione generale `tpad.conf`, localizzando una linea del tipo:

   ```
   # tidy(1) path (empty if you don't have it)
   variable tidy {}
   ```

   Qui inserire il percorso dell'eseguibile `tidy`. Assumendo che `tidy(1)` si trovi nel percorso di ricerca di default, sarà sufficiente aggiungere solo il nome del comando:

   ```
   # tidy(1) path (empty if you don't have it)
   variable tidy {tidy}
   ```

3. ripetere il passo 2 per ciascun file di configurazione per utente `~/.tpadrc`, se l'utente è interessato ad utilizzare tidy in tpad

Limitazioni

- Il plugin è stato testato solamente con la versione di HTML TIDY rilasciata il 4 Agosto 2000.

- Utilizzando l'interfaccia del plugin è possibile impostare solo un sottoinsieme delle numerose opzioni di Tidy. Tutte le altre devono essere impostate tramite file di configurazione esterni. Consultare la documentazione di Tidy a proposito.

- L'opzione `-slides` per la generazione di slide in formato HTML è stata omessa perchè causa una eccezione di tipo SIGSEGV sul mio sistema.
- Dal momento che il plugin utilizza l'opzione `-q` di Tidy, fatta apposta per facilitare il parsing dell'output alle applicazioni che si interfacciano con Tidy da riga di comando, una dichiarazione di DOCTYPE sbagliata non viene segnalata con un messaggio di avvertimento nella modalità "Valida". Comunque il modo "Valida e correggi" è in grado di aggiustare la dichiarazione DOCTYPE scorretta. Se utilizzate solo il modo "Valida" dovete perciò ricordarvi di inserire una dichiarazione DOCTYPE appropriata manualmente.

2.6 Creazione di file di log

Per creare un file di registrazioni manuale, basta partire da un testo vuoto e inserire come prima linea esattamente la seguente:

```
.LOG
```

Poi si salva il documento e lo si riapre ogni volta che si deve inserire una nuova registrazione.

Ogni volta che si apre un file che inizia con una linea `.LOG`, tpad inserirà automaticamente alla fine del testo la data e l'ora corrente nel formato della lingua corrente di sistema (si tratta della stessa funzione operata dall'opzione "Ora/Data" del menu "Edit").

Questo comportamento risulta molto comodo per scrivere il proprio diario con tpad.

Per disabilitare questa funzionalità, che è attiva per default, si veda la descrizione della variabile `logfac` nel [§ 2.1].

Se tpad si trova nel modo a sola lettura [§ 2.4.1] l'unico effetto che si ottiene impostando la variabile `logfac` a `yes` è che la vista scorre fino alla fine del file ogni volta che il file di log viene aperto. In questo caso non viene inserita alcuna data.

Se si vuole evitare lo scorrimento fino alla fine del testo quando si apre un file di log si può utilizzare l'opzione di riga di comando "vai a" [§ 2.4]. Per esempio:

```
$ tpad +1 mylog.log
$ tview +1 diary
```

Capitolo 3

Documentazione per il programmatore

3.1 Utilizzare tpad in altri programmi

3.1.1 Introduzione

A partire dalla versione 1.3 tpad è stato pacchettato sotto forma di una libreria Tcl per facilitare il suo riuso in altri programmi. In molti casi sarà sufficiente richiamare tpad come un sottoprocesso utilizzando il comando `exec(n)` e la sua interfaccia a riga di comando [§ 2.4]. Ma quando occorre operare un maggiore controllo sull'istanza di tpad e, per ragioni di sicurezza, non si vuole usare il meccanismo di comunicazione tra applicazioni offerto da Tcl tramite il comando send(n) [§ 1.6, p. 17], è necessario istanziare e controllare tpad attraverso la sua libreria di comandi.

Tpad non è una libreria *orientata agli oggetti* (*object oriented*), principalmente perchè non ho voluto aggiungere [incr Tcl] tra le dipendenze. Normalmente infatti Tcl/Tk è già presente in moltissime installazioni di Unix, ma non altrettanto si può dire della sua estensione per l'OOP.

Ho quindi optato per una architettura più semplice del tipo *basata sugli oggetti* (*object based*) [§ 1.5, p. 12], senza peraltro rinunciare a quei fattori di qualità del software che sono maggiormente influenzati dall'approccio OOP.

Grazie a varie fattori, quali la natura dinamica di Tcl, le funzionalità di strutturazione dei programmi che fornisce (comandi, namespace e package) e di comunicazione tra applicazioni (meccanismo del send), e all'adesione alle convenzioni di Unix per facilitare il riuso (interfacce da riga di comando), il risultato finale è che tpad è parimenti estendibile e riusabile pure senza [incr Tcl] e anche le qualità interne di strutturazione, modularità e comprensibilità del codice sono state curate.

Come Tcl/Tk, anche tpad aspira a raggiungere il difficile obiettivo di essere software di elevate qualità interne ed esterne e nel contempo richiedere un basso *footprint* in termini di risorse hardware e software, in modo da poter essere usato anche in dispositivi mobili con capacità limitate, specialmente in vista della sua estensione come sistema di word-processing di cui si è parlato nella [§ 1.8].

3.1.2 La semplice API offerta da tpad

Utilizzo di base

La prima cosa da fare per poter utilizzare il package tpad è di richiederne l'uso a Tcl:

```
package require tpad
```

Questo comando cercherà di individuare automaticamente la più recente versione della libreria tpad disponibile. Viene ritornato il numero di versione oppure un errore se nessuna versione di tpad è stata installata. Vedere la pagina di manuale di `package(n)` oppure la [§ 3.1.2] per maggiori dettagli su come richiedere dei particolari numeri di versione quando più versioni di tpad sono installate sulla stessa macchina.

Tpad utilizza la funzionalità di auto-loading di Tcl. Questo significa che richiedere la libreria come fatto sopra non implica che il file principale che la implementa (`tpad.tcl`) venga caricato immediatamente. Invece il caricamento viene ritardato finché non si utilizza per la prima volta uno dei comandi esportati dalla libreria. In altri termini se solo si richiede l'uso della libreria tpad non si ha nessun overhead nel programma dell'utente. La libreria verrà caricata solo quando viene effettivamente utilizzata. Questa misura riduce i tempi di avvio dei programmi interattivi.

Opzionalmente dopo aver richiesto l'uso della libreria, si possono importare tutti i suoi comandi pubblici con:

```
namespace import ::tpad::*
```

Il comando `namespace(n) import` consente di accedere a tutti i comandi di tpad senza dover utilizzare il prefisso di namespace `::tpad::`, ma può produrre degli errori a causa delle collisioni tra nomi. Questo avviene ogni volta che si importa un comando da un altro package che ha lo stesso nome di uno dei comandi esportati da tpad. Nel seguito continueremo ad utilizzare il prefisso `::tpad::` prima di ciascun comando di tpad, assumendo che nessuno di essi sia stato importato. Per ridurre il rischio di collisioni infatti si possono importare i comandi di tpad anche individualmente. Per esempio, per importare solamente i comandi `tpad` e `display` utilizzare:

```
namespace import ::tpad::tpad ::tpad::display
```

Per una lista sempre aggiornata dei comandi esportati vedere il file di libreria `tpad.tcl` e cercare il comando `namespace export` al suo interno. Questo file è stato riportanto anche nell'[Appendice A.1].

Il prossimo passo consiste nel creare una istanza di tpad. Ciò deve essere fatto prima di chiamare qualsiasi altro comando (fatta eccezione per i comandi `::tpad::cmdHelp` e `::tpad::whatIs` in quanto sono gli unici che non richiedono un pathname come argomento). Una istanza di tpad può essere visualizzata nella finestra principale dell'applicazione (pathname `.`) oppure in una finestra secondaria (detta anche un *toplevel widget*). Utilizzare il comando `toplevel(n)` per creare e configurare una finestra secondaria da utilizzare per tpad. Poi far uso del comando `::tpad::tpad` per creare e inizializzare una nuova istanza di tpad. Il primo argomento è il pathname o percorso della finestra da usare. Il secondo deve essere `tpad` per l'editor oppure `tview` se si vuole utilizzare tpad come un visualizzatore (paginatore). Come osservato sopra, questo comando provocherà il caricamento automatico della libreria tpad se non è già stata interpretata prima. Due esempi:

```
::tpad::tpad . tpad

::tpad::tpad .txt tview
toplevel .txt
```

Il terzo argomento del comando `::tpad::tpad` è opzionale e deve essere la lista delle opzioni con la stessa sintassi utilizzata dalla riga di comando. Vedere a proposito la [§ 2.4] oppure la manpage `tpad(1)`. Esempio:

```
::tpad::tpad . tpad +10 file1 /path/file2
```

Per poter passare proprio le opzioni ricevute dalla riga di comando dal programma che incorpora tpad occorre usare il comando `eval(n)`, altrimenti la varabile `$argv` verrebbe trasmessa come un argomento singolo alla procedura tpad e questo sarebbe un errore se `arvg` contiene più di un argomento oppure è vuoto:

```
eval ::tpad::tpad . $argv0 $argv
```

In questo esempio è stata utilizzata la variabile che contiene il nome del programma `argv0` per distinguere tra l'editor e il viewer.

Infine per visualizzare tpad si chiama la procedura `display`, ripassandole il pathname:

```
::tpad::display .
```

Il pathname (abbreviato in `pn` nelle liste di argomenti) va passato come primo parametro a quasi tutti i comandi esportati dalla libreria tpad. Va pensato come se fosse l'oggetto corrente o il puntatore `this` nella programmazione orientata agli oggetti. Si possono avere più istanze di tpad purché ciascuna usi un diverso pathname. Il pathname unico permette appunto di distinguere tra le varie istanze. Esempio:

```
toplevel .tpad1
toplevel .tpad2
::tpad::tpad .tpad1 tpad
::tpad::tpad .tpad2 tpad
::tpad::display .tpad1
::tpad::display .tpad2
```

Ecco ora degli esempi d'uso di ulteriori comandi che sono disponibili. Il primo ritorna il numero complessivo di linee del testo corrente, per la istanza `.` (questo numero è al minimo 1, in quanto c'è sempre almeno una linea vuota nel testo):

```
::tpad::lines .
```

Il comando seguente produce lo stesso effetto dell'opzione "File/Salva" per l'istanza `.tpad2`:

```
::tpad::save .tpad2
```

Se preferite non utilizzare file temporanei per scambiare il testo che deve essere modificato tra tpad e l'applicazione che lo incorpora, tpad supporta anche l'editing diretto di una variabile di memoria. Si può scegliere se rendere la finestra di tpad modale oppure no (vedere `grab(n)`). Si usa quindi il nome della variabile come ulteriore argomento della procedura `display`. Occorre anche utilizzare `trace(n)` per registrare una funzione di callback che verrà invocata quando la variabile che contiene il testo viene modificata (questo avviene quando l'utente esce da tpad confermando il salvataggio del buffer). Il monitoraggio della variabile con `trace` rappresenta un modo semplice ed efficiente per sapere se la variabile è stata cambiata. L'esempio seguente mette in pratica quanto detto:

```
set txt {This is
the txt to edit.}

trace variable txt w txtChanged

proc txtChanged {variableName elementName operation} {
  # remember that this proc executes at the same
  # stack level as the access to the variable txt

  puts "txt variable modified.\nThe new value is:\n$::txt"
}

toplevel .ed
grab .ed ;# use a modal edit window (optional)
```

```
::tpad::tpad .ed tpad
set ::tpad::titlefmt {my thoughts}
::tpad::display .ed txt

# ...
```

Nota sulla chiusura delle finestre di tpad: quando per tpad si utilizza un toplevel widget diverso dalla finestra principale `.` (ad es. si usa `.tpad`), occorre ricordarsi di chiamare esplicitamente la procedura `quit` di tpad all'interno del gestore degli eventi della finestra genitore (rispetto a quella che contiene tpad). Se dimenticate di farlo, quando la finestra genitore viene chiusa *prima* di quella figlia con tpad, anche quest'ultima verrà distrutta, senza dare nessuna opportunità all'utente di salvare eventuali modifiche del testo. L'esempio seguente mostra come fare per testare se la finestra di tpad è ancora aperta e fornire quindi un modo sicuro di distruggerla, quando necessario:

```
toplevel .tpad
wm protocol . WM_DELETE_WINDOW {
  # this is the parent closing handler
  # ...

  if {[namespace exists ::tpad::.tpad]} {
    ::tpad::quit .tpad ;# clean quit from tpad
  }

  # ...
  # tipically you will have a "destroy ." as the last command
}

eval ::tpad::tpad .tpad
::tpad::display .tpad
```

Utilizzo avanzato

Come abbiamo visto nella [§ 3.1.2], dopo aver creato una istanza di tpad, si usa la procedura `display` per visualizzare l'intera applicazione.

È questo il momento in cui vengono definiti tutti i menu, le combinazioni di tasti, le funzioni di callback e vengono pacchettati i widget che compongono la GUI. Se intendete riutilizzare il codice di tpad nella costruzione di un editor integrato per altri programmi (ad es. un programma di posta elettronica, un debugger, una IDE, ecc.) vorrete probabilmente usare una vostra versione modificata della procedura `display` ed eventualmente di altre callback (oppure vorrete ridefinire completamente alcune delle callback).

La procedura `display` è definita in un file separato `display.tcl` nella sottodirectory `lib` di tpad proprio allo scopo di poterla ridefinire in modo più efficiente a runtime[1].

Infatti definendo una versione personalizzata del comando `::tpad::display` prima di invocarlo o importarlo tramite `namespace import`, la versione di default non verrà mai caricata. Al suo posto verrà caricata la versione da voi fornita, evitando l'inutile overhead del caricamento della versione di default.

Per definire la vostra versione particolarizzata di `::tpad::display`, è più semplice copiare e modifica la versione di default inclusa in tpad piuttosto che cominciare da zero. Attenzione che se si copia da `display.tcl` occorre cancellare la linea `package provide`.

Si prega di non provare a modificare direttamente il file `display.tcl` fornito con tpad, in quanto si potrebbe produrre qualche malfunzionamento in altri programmi nello stesso sistema che dipendono da tpad e utilizzano la versione di default di questo file, incluso lo script di avvio dell'editor di esempio `tpad`.

Infine notare che il file sorgente `display.tcl` che definisce l'aspetto complessivo di tpad può cambiare da una versione all'altra. Non c'è nessuna garanzia che una versione più vecchia di questo file funzioni con una versione più recente della libreria di tpad. Perciò se ridefinite l'aspetto di tpad, assicuratevi di richiedere nel vostro codice la versione esatta della libreria tpad che state utilizzando, tramite l'argomento `-exact` del comando `package require`:

```
package require tpad -exact 1.3
```

Questo paragrafo è stato intitolato "Utilizzo avanzato" perchè il riuso di tpad in questo modo richiede una maggiore conoscenza di programmazione in Tcl.

Questo è l'inevitabile prezzo da pagare per poter personalizzare completamente l'aspetto e il comportamento del proprio editor. È pertanto richiesto lo studio del file di visualizzazione di default fornito con tpad [Appendice A.2] prima di cominciare a codificare la propria versione.

[1]Si ricordi che Tcl è un linguaggio dinamico e consente la ridefinizione dei comandi a runtime, cosa non possibile nei linguaggi puramente compilati.

Appendice A

Alcuni sorgenti

In questa Appendice per brevità non è stato riportato per esteso tutto il codice che compone la libreria tpad.

A scopo esemplificativo, in [A.2 e A.3] vengono presentati (in versione completa) due file sorgente di cui si è parlato diffusamente nei [Capitoli 1 e 3].

Del file principale che implementa la libreria tpad invece, in [A.1], si è evidenziata solo la struttura complessiva, trascurando del tutto i comandi privati che non fanno parte dell'interfaccia della libreria e, per quanto riguarda i comandi pubblici esportati, omettendo i corpi che li implementano e lasciando solamente l'intestazione che li documenta, secondo le convenzioni della Guida di Stile Tcl [12].

I sorgenti completi e il log delle modifiche apportate in ciascuna versione del programma sono reperibili nel sito web di tpad [4].

A.1 `tpad.tcl`

```
# tpad.tcl --
#
#   This is the primary file in the tpad library and implements
#   the Tcl code for both exported and internal commands.
#
# Copyright (C) 2005  Antonio Bonifati
#
# See the file "license.terms" for information on usage and
# redistribution of this file, and for a DISCLAIMER OF ALL WARRANTIES.
#

...

package provide tpad 1.3
```

```
namespace eval ::tpad {
  # Init commands: note that the tpad command must be called
  # *before* any other exported command and the default
  # version of display is defined in display.tcl.
  namespace export tpad

  # Public editor menu commands.
  namespace export new next tpadOpen save saveAs printSetup print quit \
    undoRedo undo redo cut copy paste tpadDelete findReplace findNext \
    goTo selectAll insertDate insertFile toggleToolBar toggleWordWrap \
    tpadFont ioEncodings toggleStatusBar toggleAsciiTable toggleHtmlTidy \
    toggleTextFormatter toggleXmlwf help tip about

  # Public useful and safe commands (not menu commands).
  namespace export lineNo lines goToLine updateStatusBar whatIs cmdHelp \
    emptyOpenRecent openRecentLength getInputEnc getOutputEnc fortune
}

## Private commands.

...

## Public useful and safe commands (not menu commands).

# tpad::getInputEnc --
#
#   Tell what encoding tpad is currently used when reading files.
#   This is one of the elements of the list returned by the
#   [encoding names] command.
#
# Arguments:
#   pn  toplevel edit window.
# Results:
#   Returns the current input encoding.

...

# tpad::getOutputEnc --
#
#   Similar to getInputEnc, but the encoding
#   is that used when writing files.
#
# Arguments:
```

```
#   pn  toplevel edit window.
# Results:
#   Returns the current output encoding.

...

# tpad::fortune --
#
#   Return a random, hopefully useful, tip.
#
# Returns:
#   Returns a tip about using tpad as a multiline string.

...

# tpad::lineNo --
#
#   Given an index in the text, returns the corresponding line number.
#   Lines are numbered from 1 for consistency with other UNIX programs
#   that use this numbering scheme.
#
# Arguments:
#   pn  toplevel edit window.
#   index   index in the text.
# Results:
#   Returns the line number corresponding to index.

...

# tpad::lines --
#
#   Find out how many lines the text currently has. An empty text has 1
#   line, not 0, because there is always a newline as the last character.
#
# Arguments:
#   pn  toplevel edit window.
# Results:
#   Returns the total number of lines in the text.

...

# tpad::goToLine --
#
```

```
#   If row is a valid line number in the text, set the insertion
#   cursor to the specified row:col position and adjust the view.
#
# Arguments:
#   pn  toplevel edit window.
#   row row number. From 1.
#   col (optional) column number. From 0. Defaults to 0,
#       which means the first character within a line.
# Returns:
#   Returns a boolean yes if the cursor positioning was successful,
#   otherwise pops up an error message window and returns no.

...

# tpad::updateStatusBar --
#
#   Update the status bar. Invoked every time a key or the left mouse
#   button is pressed or released, after a successful search, a cursor
#   repositioning, a paste operation and so on.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

...

# tpad::whatIs --
#
#   Return program brief description with version number.
#
# Returns:
#   Returns a localized string describing tpad.

...

# tpad::cmdHelp --
#
#   Return program brief description as tpad::whatIs
#   plus command line syntax.
#
# Returns:
#   Returns a localized string describing tpad
```

```
#   and how to use it from the command line.

...

## End of public useful commands (not menu commands).


## Public editor menu commands.

# tpad::new --
#
#   Switch to edit a new empty text.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

...

# tpad::next --
#
#   Arranges for editing the next file in sequence if there's one,
#   otherwise simply disables the Next option.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns a boolean yes if a next file in sequence is
#   actually loaded or created as new, no otherwise.

...

# tpad::tpadOpen --
#
#   Open a new file for editing.
#
# Arguments:
#   pn      toplevel edit window.
#   newFilename (optional) name of the file to edit. Defaults to an
#           empty string, which means to ask the user.
# Returns:
#   Returns an empty string.
```

```
...

# tpad::save --
#
#   Write the file back. Ask the user for the name of the file is not
#   defined yet. Also ask for confirmation if the file is read-only.
#   or appears to be more recently than this copy.
#
#   This proc no more does nothing if the text hasn't been modified from
#   last saved version, because one may want to save the text to convert
#   it to another encoding, without modifying it and also because these
#   optimizations are automatic in modern OS's.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns a boolean yes if the file was saved, no if the user has
#   cancelled the operation.

...

# tpad::saveAs --
#
#   Save the current copy of the text on a file with a new name.
#   This will be the new file to be edited.
#
# Arguments:
#   pn      toplevel edit window.
#   newFilename (optional) new name of the file. Defaults to
#           an empty string, which means to ask the user.
# Returns:
#   Returns a boolean yes if the user has chosen to save,
#   no if the user has cancelled the operation.

...

# tpad::tpadFont --
#
#   Display a dialog allowing the user to change
#   the editor and ascii table plugin font.
#
# Arguments:
```

```
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

...

# tpad::ioEncodings --
#
#   Display a modal window allowing the user to set the encodings used
#   to read/write files. tpad will convert from UTF-8 (used internally
#   by Tcl) to the specified encoding whenever reads or writes files.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

...

# tpad::printSetup --
#
#   Open the modal Print setup dialog box.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

...

# tpad::print --
#
#   Print the text edited so far, piping it into lpr(1) or similar.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

...

# tpad::quit --
#
```

```
#   Quit from from the application, if the user confirms.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

...

# tpad::undoRedo --
#
#   Implement one level undo/redo, i.e. toggle
#   between undo and redo whichever is applicable.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

...

# tpad::undo --
#
#   Undo the last edit action. Do nothing when the undo stack is empty.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

...

# tpad::redo --
#
#   Reapply the last undone edits provided no other edits were done
#   since then. Do nothing when the redo stack is empty.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

...
```

```
# tpad::cut --
#
#   Copy the selection in the widget to the clipboard and delete the
#   selection. If there is no selection in the widget do nothing.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

...

# tpad::copy --
#
#   Copy the selection in the widget to the clipboard,
#   if there is a selection.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

...

# tpad::paste --
#
#   Inserts the contents of the clipboard at the position of the
#   insertion cursor. Pasting over a selection replaces it.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

...

# tpad::tpadDelete --
#
#   Delete the selection, if there is one in the text.
#
# Arguments:
#   pn  toplevel edit window.
```

```
# Returns:
#   Returns an empty string.

...

# tpad::findReplace --
#
#   Opens the find or find&replace dialog box if it isn't already opened.
#
# Arguments:
#   pn  toplevel edit window.
#   replace (optional) boolean. If it is yes the replace features will be
#       included. Defaults to no, which means allow only searches,
#       not string replacement.
# Returns:
#   Returns an empty string.

...

# tpad::findNext --
#
#   Find the next occurrence of the pattern. If the pattern is empty
#   (not defined) or the Find window is opened, invoke findReplace instead.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Same as FindPattern if the search had place,
#   otherwise returns an empty string.

...

# tpad::goTo --
#
#   Opens the modal Goto line dialog box.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

...
```

```
# tpad::selectAll --
#
#   Select the entire text.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

...

# tpad::insertDate --
#
#   Insert locale specific date and time just before the insertion cursor.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

...

# tpad::insertFile --
#
#   Insert a file just below the cursor.
#
# Arguments:
#   pn      toplevel edit window.
#   insFilename (optional) name of the file to insert. Defaults to an
#           empty string, which means to ask the user.
# Returns:
#   Returns an empty string.

...

# tpad::toggleToolBar --
#
#   Display/hide the toolbar.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.
```

```
...

# tpad::toggleWordWrap --
#
#   Enable/disable word wrapping mode.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

...

# tpad::toggleAsciiTable --
#
#   Display/hide the Ascii Table plugin.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

...

# tpad::toggleHtmlTidy --
#
#   Display/hide the HTML Tidy plugin.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

...

# tpad::toggleTextFormatter --
#
#   Display/hide the Text Formatter plugin.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
```

```
#   Returns an empty string.

...

# tpad::toggleXmlwf --
#
#   Display/hide the Xmlwf plugin.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

...

# tpad::toggleStatusBar --
#
#   Display/hide the status bar.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

...

# tpad::tip --
#
#   Print a random tip about using tpad.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

...

# tpad::emptyOpenRecent --
#
#   Empty the open recent file list in memory.
#
# Arguments:
#   pn  toplevel edit window.
```

```
# Returns:
#   Returns an empty string.

...

# tpad::openRecentLength --
#
#   Compute the length of the open recent file list.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns a decimal string giving the number of elements in the list.

...

# tpad::help --
#
#   Fire up the browser to read tpad documentation.
#
# Returns:
#   Returns an empty string.

...

# tpad::about --
#
#   Pop up the modal about message window.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

...

## End of public editor menu commands.


## Init commands.

# tpad::tpad
#
```

```
#   Given pathname (pn), a toplevel window widget
#   created by toplevel(n) or the main window (.),
#   build up an instance of the tpad editor inside it.
#
# Arguments:
#   name    name of the executable. Must be tpad for the editor,
#       tview for the viewer.
#   args    list of options with the same syntax used on the command line.
#       Indeed it can be set to $argv or even left empty.
# Returns:
#   Returns an empty string.

...

## End of init commands.


...
```

A.2 display.tcl

```
# display.tcl --
#
#   This is a secondary file in the tpad library and implements the
#   Tcl code for defining the default interface of the tpad
#   application.
#
# Copyright (C) 2005  Antonio Bonifati
#
# See the file "license.terms" for information on usage and
# redistribution of this file, and for a DISCLAIMER OF ALL WARRANTIES.
#

package provide tpad 1.3

namespace eval ::tpad {
  namespace export display
}

# tpad::display --
#
#   Display a Notepad clone application in a window and start it.
#   If no additional argument is present, the editor will operate on a
```

```
#   file (as a standalone and fully independant application),
#   otherwise the additional argument must be the name of a variable.
#   If the user confirmed the editing, this variable will be set upon
#   exiting with a copy of the edited text, otherwise it will be
#   leaved unmodified. The variable initial value will be used to
#   initialize the text to be edited. Use trace(n) to execute a
#   callback every time $stringName is modified.
#
# Arguments:
#   pn      toplevel edit window.
#   stringName  (optional) name of a string variable to edit. Defaults
#           to an empty string, which means to configure the editor to
#           operate on a file instead of a variable.
# Results:
#   Returns an empty string.

proc ::tpad::display {pn {stringName {}}} {
  foreach v {p argv0 argv edit line} {
    variable $pn\::$v
  }

  if {$stringName!={}} {
    variable $pn\::var $stringName
  } else {
    variable $pn\::var {}
  }

  tk_setPalette $::tpad::palette

  # sets window manager options
  wm title $pn [format $::tpad::titlefmt [file tail $argv0] \
    [mc Untitled]]
  wm iconname $pn [file tail $argv0]
  wm command $pn [concat $argv0 $argv]
  wm group $pn $pn
  # catch the kill of the window manager
  wm protocol $pn WM_DELETE_WINDOW [namespace code "quit $pn"]
  # end of window manager options

  # builds the menus
  set m $p.menubar
  menu $m
  AmpMenu $m add cascade -menu $m.file -lab [mc &File]
```

```
AmpMenu $m add cascade -menu $m.edit -lab [mc &Edit]
AmpMenu $m add cascade -menu $m.format -lab [mc F&ormat]
AmpMenu $m add cascade -menu $m.view -lab [mc &View]
AmpMenu $m add cascade -menu $m.help -lab [mc &Help]

set w $m.file
menu $w -tearoff $::tpad::tearoff
if {$var!={}} {
  AmpMenu $w add comm -lab [mc &Reset] \
    -comm [namespace code "new $pn"]
} elseif {$edit} {
  AmpMenu $w add comm -lab [mc &New] -accel Ctrl+N \
    -comm [namespace code "new $pn"]
} else {
  AmpMenu $w add comm -lab [mc &Clear] -accel Ctrl+N \
    -comm [namespace code "new $pn"]
}
AmpMenu $w add comm -lab [mc Ne&xt] \
  -comm [namespace code "next $pn"]
AmpMenu $w add comm -lab [mc &Open...] -accel Ctrl+F12 \
  -comm [namespace code "tpadOpen $pn"]

if {$::tpad::maxopenrec} {
  AmpMenu $w add cascade -lab [mc "Op&en Recent"] -menu $w.recent
  menu $w.recent -tearoff $::tpad::tearoff
  set entries [LoadOpenRecent $pn]
  $w.recent add separator
  $w.recent add comm -lab [mc "empty this list"] \
    -comm [namespace code "emptyOpenRecent $pn"]
  if {$entries==0} {
    $w.recent entryconf end -state disabled
  }
}

if {$edit} {
  if {$var=={}} {  ;# it is safe to remove this test if you want to
    AmpMenu $w add comm -lab [mc &Save] -accel Shift+F12 \
      -comm [namespace code "save $pn"]
  }
  AmpMenu $w add comm -lab [mc "Sa&ve as..."] \
    -comm [namespace code "saveAs $pn"]
}
$w add separator
```

```
AmpMenu $w add comm -lab [mc "Pr&int setup..."] \
  -comm [namespace code "printSetup $pn"]
AmpMenu $w add comm -lab [mc &Print] -accel Ctrl+Shift+F12 \
  -comm [namespace code "print $pn"]
$w add separator
AmpMenu $w add comm -lab [mc &Quit] \
  -comm [namespace code "quit $pn"]

set w $m.edit
menu $w -tearoff $::tpad::tearoff \
  -postcommand [namespace code "SetStateFromClipboard $pn $w"]
if {$edit} {
  if {$::tpad::maxundo!=1} {
    AmpMenu $w add comm -lab [mc &Undo] -accel Ctrl+Z \
      -comm [namespace code "undo $pn"]
    AmpMenu $w add comm -lab [mc &Redo] -accel Ctrl+R \
      -comm [namespace code "redo $pn"]
  } else {
    AmpMenu $w add comm -lab [mc &Undo] -accel Ctrl+Z \
      -comm [namespace code "undoRedo $pn"]
  }
  $w add separator
  AmpMenu $w add comm -lab [mc &Cut] -accel Ctrl+X \
    -comm [namespace code "cut $pn"]
}
AmpMenu $w add comm -lab [mc C&opy] -accel Ctrl+C \
  -comm [namespace code "copy $pn"]
if {$edit} {
  AmpMenu $w add comm -lab [mc &Paste] -accel Ctrl+V \
    -comm [namespace code "paste $pn"]
  AmpMenu $w add comm -lab [mc &Delete] -accel Del \
    -comm [namespace code "tpadDelete $pn"]
}
$w add separator
AmpMenu $w add comm -lab [mc &Find...] -accel Ctrl+F \
  -comm [namespace code "findReplace $pn"]
AmpMenu $w add comm -lab [mc "Find &next"] -accel F3 \
  -comm [namespace code "findNext $pn"]
if {$edit} {
  AmpMenu $w add comm -lab [mc R&eplace...] -accel Ctrl+H \
    -comm [namespace code "findReplace $pn yes"]
}
AmpMenu $w add comm -lab [mc "&Goto line..."] -accel Ctrl+G \
```

```
  -comm [namespace code "goTo $pn"]
$w add separator
AmpMenu $w add comm -lab [mc "&Select all"] \
  -accel Ctrl+5(kp) -comm [namespace code "selectAll $pn"]
if {$edit} {
  AmpMenu $w add comm -lab [mc Date/&time] -accel F5 \
    -comm [namespace code "insertDate $pn"]
  AmpMenu $w add comm -lab [mc "&Insert file..."] \
    -comm [namespace code "insertFile $pn"]
}

set w $m.format
menu $w -tearoff $::tpad::tearoff
AmpMenu $w add checkbutton -var ::tpad::$pn\::wrap \
  -onv word -offv none -lab [mc "&Word wrap"] \
  -comm [namespace code "toggleWordWrap $pn"]
if {$::tpad::xfontsel!={}} {
  AmpMenu $w add comm -lab [mc "&Font..."] \
    -comm [namespace code "tpadFont $pn"]
}
AmpMenu $w add comm -lab [mc "&I/O Encodings"] \
  -comm [namespace code "ioEncodings $pn"]

set w $m.view
menu $w -tearoff $::tpad::tearoff
AmpMenu $w add checkbutton -var ::tpad::$pn\::tool \
  -onv yes -offv no -lab [mc "&Tool bar"] \
  -comm [namespace code "toggleToolBar $pn"]
AmpMenu $w add checkbutton -var ::tpad::$pn\::status \
  -onv yes -offv no -lab [mc "&Status bar"] \
  -comm [namespace code "toggleStatusBar $pn"]
$w add separator
AmpMenu $w add checkbutton -var ::tpad::$pn\::ascii \
  -onv yes -offv no -lab [mc "&Ascii table"] \
  -comm [namespace code "toggleAsciiTable $pn"]
if {$::tpad::tidy!={}} {
  AmpMenu $w add checkbutton -var ::tpad::$pn\::ht \
    -onv yes -offv no -lab [mc "&HTML Tidy"] \
    -comm [namespace code "toggleHtmlTidy $pn"]
}
if {$edit && $::tpad::fmtkind!={none}} {
  AmpMenu $w add checkbutton -var ::tpad::$pn\::format \
    -onv yes -offv no -lab [mc "Text &formatter"] \
```

```
    -comm [namespace code "toggleTextFormatter $pn"]
}
if {$::tpad::xmlwf!={}} {
  AmpMenu $w add checkbutton -var ::tpad::$pn\::wf \
    -onv yes -offv no -lab [mc "&XML wf checker"] \
    -comm [namespace code "toggleXmlwf $pn"]
}

set w $m.help
menu $w -tearoff $::tpad::tearoff
AmpMenu $w add comm -lab [mc &Help...] -comm [namespace code help]
AmpMenu $w add comm -lab [mc &Tips...] \
  -comm [namespace code "tip $pn"]
AmpMenu $w add comm -lab [mc &About...] \
  -comm [namespace code "about $pn"]

set w $p.rightmenu
menu $w -tearoff $::tpad::tearoff \
  -postcommand [namespace code "SetStateFromClipboard $pn $w"]
if {$edit} {
  if {$::tpad::maxundo!=1} {
    AmpMenu $w add comm -lab [mc &Undo] \
      -comm [namespace code "undo $pn"]
    AmpMenu $w add comm -lab [mc &Redo] \
      -comm [namespace code "redo $pn"]
  } else {
    AmpMenu $w add comm -lab [mc &Undo] \
      -comm [namespace code "undoRedo $pn"]
  }
  $w add separator
  AmpMenu $w add comm -lab [mc &Cut] -comm [namespace code "cut $pn"]
}
AmpMenu $w add comm -lab [mc C&opy] -comm [namespace code "copy $pn"]
if {$edit} {
  AmpMenu $w add comm -lab [mc &Paste] \
    -comm [namespace code "paste $pn"]
  AmpMenu $w add comm -lab [mc &Delete] \
    -comm [namespace code "tpadDelete $pn"]
}
$w add separator
AmpMenu $w add comm -lab [mc "&Select all"] \
  -comm [namespace code "selectAll $pn"]
# end of menus' definition
```

```
  # define the widgets
  set w $p.toolBar
  frame $w -bd 2 -relief raised -padx 5 -pady 3
  if {$edit} {
    ButtonRow $pn $w.file {new open save} {new tpadOpen save}
    ButtonRow $pn $w.print {print preview} {print}
    ButtonRow $pn $w.find {find} {findNext}
    if {$::tpad::maxundo!=1} {
      ButtonRow $pn $w.clip {cut copy paste undo redo} \
        {cut copy paste undo redo}
    } else {
      ButtonRow $pn $w.clip {cut copy paste undo} \
        {cut copy paste undoRedo}
    }
    ButtonRow $pn $w.datetime {datetime} {insertDate}
    pack $w.file $w.print $w.find $w.clip $w.datetime -side left \
      -padx 5
  } else {
    ButtonRow $pn $w.file {new open} {new tpadOpen}
    ButtonRow $pn $w.print {print preview} {print}
    ButtonRow $pn $w.find {find} {findNext}
    ButtonRow $pn $w.clip {copy} {copy}
    pack $w.file $w.print $w.find $w.clip -side left -padx 5
  }

  set w $p.text
  text $w -font $::tpad::editfont -fg $::tpad::fg -bg $::tpad::bg \
    -wrap none -setgrid yes -undo $edit -maxundo $::tpad::maxundo \
    -insertofftime $::tpad::insertofftime \
    -insertontime $::tpad::insertontime \
    -yscroll [namespace code "UpdateOffset $pn"] \
    -xscroll [namespace code "$p.hscroll set"]
  $w tag configure sel -foreground $::tpad::selfg \
    -background $::tpad::selbg
  $w tag configure dupsel -foreground $::tpad::selfg \
    -background $::tpad::selbg
  scrollbar $p.vscroll -comm [namespace code "$w yview"] \
    -width $::tpad::scrollwidth
  scrollbar $p.hscroll -comm [namespace code "$w xview"] \
    -orient horizontal -width $::tpad::scrollwidth

  set w $p.statusBar
```

```
frame $w -pady 2
label $w.filename -anchor w -bd 1 -font $::tpad::statusfont \
  -text [mc Untitled]
label $w.modified -relief sunken -width [mcmax Modified Readonly] \
  -bd 1 -font $::tpad::statusfont
label $w.lines -relief sunken -text 1 \
  -width 6 -bd 1 -font $::tpad::statusfont
label $w.fraction -relief sunken -width 8 -bd 1 \
  -font $::tpad::statusfont
label $w.coords -relief sunken -text 1,1 \
  -width 8 -bd 1 -font $::tpad::statusfont
pack $w.filename -side left -exp yes -fill x -padx 2
pack $w.modified $w.lines $w.fraction $w.coords -side left -padx 2
# end of widgets' definition

# install or disable X event handlers
if {$var=={}} {
  bind $pn <Control-Key-n> [namespace code "new $pn"]
  # this should be preferable than a "bind Text <Control-Key-n> {}"
  bind $p.text <Control-Key-n> [namespace code "new $pn; break"]
}
bind $pn <Control-Key-F12> [namespace code "tpadOpen $pn"]
bind $pn <Control-Shift-Key-F12> [namespace code "print $pn"]
catch {bind $pn <Control-Shift-Key-XF86_Switch_VT_12> [
  namespace code "print $pn"
]}
catch {bind $p.text <Control-Key-KP_Begin> \
  [namespace code "selectAll $pn"]}
bind $p.text <Control-Key-Clear> [namespace code "selectAll $pn"]
bind $p.text <Control-Key-f> \
  [namespace code "findReplace $pn; break"]
bind $p.text <Control-Key-h> \
  [namespace code "findReplace $pn yes; break"]
bind $p.text <Control-Key-g> [namespace code "goTo $pn"]
bind $p.text <Key-F3> [namespace code "findNext $pn"]
bind $p.text <<Modified>> [namespace code "Modified $pn"]
bind $p.text <KeyRelease> [namespace code "
  updateStatusBar $pn
  SetStateFromSelection $pn
"]
bind $p.text <ButtonRelease> [namespace code "updateStatusBar $pn"]
bind $p.text <KeyPress> [namespace code "updateStatusBar $pn"]
bind $p.text <ButtonPress> [namespace code "updateStatusBar $pn"]
```

```
bind $p.text <ButtonPress-3> \
  [namespace code "tk_popup $p.rightmenu %X %Y"]
bind $p.text <<Selection>> [namespace code "
  DuplicateSelection $pn
  SetStateFromSelection $pn
"]
if {$edit} {
  if {$var=={}} {
    bind $pn <Shift-Key-F12> [namespace code "save $pn"]
    catch {bind $pn <Shift-Key-XF86_Switch_VT_12> \
      [namespace code "save $pn"]}
  }
  bind $p.text <Key-F5> [namespace code "insertDate $pn"]

  # here it seems better to use break than to destroy the <<Undo>>
  # binding for all Text widgets using "bind Text <<Undo>> {}"
  bind $p.text <<Redo>> [namespace code "redo $pn; break"]
  if {$::tpad::maxundo!=1} {
    event add <<Redo>> <Control-Key-r>
    bind $p.text <<Undo>> [namespace code "undo $pn; break"]
  } else {
    bind $p.text <<Undo>> [namespace code "undoRedo $pn; break"]
  }
  InstallInterceptor $pn

  bind $p.text <<Paste>> [namespace code "paste $pn; break"]
}
# end of handlers' definition

# packing
$pn config -menu $m
grid $p.toolBar -row 0 -column 0 -columnspan 2 -sticky ew
grid $p.text -row 1 -column 0 -sticky news
grid $p.vscroll -row 1 -column 1 -sticky ns
grid $p.hscroll -row 2 -column 0 -sticky ew
grid $p.statusBar -row 3 -column 0 -columnspan 2 -sticky ew
grid rowconfigure $pn 1 -weight 1
grid columnconfigure $pn 0 -weight 1
focus $p.text
# end of packing

toggleToolBar $pn
toggleWordWrap $pn
```

```
  toggleStatusBar $pn
  toggleAsciiTable $pn
  toggleTextFormatter $pn
  toggleXmlwf $pn
  toggleHtmlTidy $pn
  SetStateFromClipboard $pn $p.menubar.edit
  SetStateFromClipboard $pn $p.rightmenu

  # this is phase 2 of command line parameters handling i.e.
  # parse remaining arguments (or multiple file sequential editing)
  if { ![next $pn] } {
    new $pn
  }
  if {$line!={}} {
    goToLine $pn $line
  }

  # make sure to display everything first so the events generated
  # below can't be ignored and the optional modal tip windows
  # always pops up over a complete interface
  update idletasks

  event generate $p.text <<Selection>>
  event generate $p.text <<Modified>>

  if {$::tpad::tip} {tip $pn}

  return
}
```

A.3 xmlwf.tcl

```
# xmlwf.tcl --
#
#   This is a secondary file in the tpad library and implements
#   the Tcl code for the Expat xmlwf plugin.
#
# Copyright (C) 2005  Antonio Bonifati
#
# See the file "license.terms" for information on usage and
# redistribution of this file, and for a DISCLAIMER OF ALL WARRANTIES.
#
```

```
package provide tpad 1.3

# make sure the msgcat namespace exists
namespace eval ::msgcat {}

namespace eval ::tpad::xmlwf {
  namespace export display

  namespace import ::msgcat::mc ::msgcat::mcmax
}

# tpad::xmlwf::InitSettings --
#
#   Initialize plugin settings from default
#   values found in tpad config file.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

proc ::tpad::xmlwf::InitSettings {pn} {
  namespace eval $pn {}

  foreach var {external parameter standalone namespace} {
    set value [set ::tpad::xmlwf::$var]

    variable $pn\::$var $value
  }

  return
}

# tpad::xmlwf::SetCheckbuttonState --
#
#   Enable/disable the "process parameter entities" checkbutton.
#   Called whenever the "parse external entities" checkbutton changes
#   state.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.
```

```
proc ::tpad::xmlwf::SetCheckbuttonState {pn} {
  variable ::tpad::$pn\::p
  variable $pn\::external

  set m $p.xmlwf
  set state [expr {$external ? {normal} : {disabled}}]
  $m.f.par configure -state $state

  return
}

# tpad::xmlwf::ProcessText --
#
#   Process text using the current plugin options.
#
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

proc ::tpad::xmlwf::ProcessText {pn} {
  variable ::tpad::$pn\::p
  variable ::tpad::$pn\::filename
  foreach v {external parameter standalone namespace} {
    variable $pn\::$v
  }

  # generate the most simple command

  set cmd [list $::tpad::xmlwf]

  set opts {}
  if {$standalone} {append opts s}
  if {$namespace} {append opts n}
  if {$external} {
    if {$parameter} {
      append opts p
    } else {
      append opts x
    }
  }
  if {$opts!={}} {lappend cmd -$opts}
```

```
  # execute it and capture output

  # temporarily change to the same directory as the file
  set oldwd [pwd]
  cd [file dirname $filename]

  set xmlwfOut [eval exec $cmd << {[$p.text get 1.0 {end - 1 char}]}]

  # return to the previous directory
  cd $oldwd

  set out [split $xmlwfOut \n]
  if {$xmlwfOut!={}} {
    set last [lindex $out end]

    # parse last line, replacing STDIN with the current filename
    # and changing column numbers so that start from 1, not 0
    scan $last {STDIN:%d:%d: %n} row col msgIdx
    set msg [string range $last $msgIdx end]
    set displayfn [expr {$filename=={} ? [mc Untitled] : $filename}]
    lset out end "[file tail $displayfn]:$row:[expr $col + 1]: $msg"

    # set the cursor where the error occurred
    ::tpad::goToLine $pn $row $col
  }

  # update the log

  set w $p.xmlwf.log.txt

  $w configure -state normal
  $w delete 1.0 end
  $w insert 1.0 [join $out \n]
  $w configure -state disabled

  return
}

# tpad::xmlwf::display --
#
#   Display the XML well-formness checker plugin non-modal dialog.
#
```

```
# Arguments:
#   pn  toplevel edit window.
# Returns:
#   Returns an empty string.

proc ::tpad::xmlwf::display {pn} {
  variable ::tpad::$pn\::p

  if {![namespace exists $pn]} {
    InitSettings $pn
  }

  set m $p.xmlwf
  toplevel $m
  wm group $m $pn
  wm title $m [mc "XML well-formness checker"]
  wm transient $m $pn
  wm resizable $m no no
  # catch the kill of the window manager
  wm protocol $m WM_DELETE_WINDOW [namespace code "
    set ::tpad::$pn\::wf no
    destroy $m
  "]

  set w $m.f
  labelframe $w -padx 1.5m -pady 1.5m
  ::tpad::AmpWidget checkbutton $w.ext \
    -text [mc "parse e&xternal entities"] \
    -var ::tpad::xmlwf::$pn\::external -onv yes -offv no \
    -comm [namespace code "SetCheckbuttonState $pn"]
  $w configure -labelwidget $w.ext
  ::tpad::AmpWidget checkbutton $w.par \
    -var ::tpad::xmlwf::$pn\::parameter \
    -text [mc "process &parameter entities"] -onv yes -offv no
  pack $w.par

  set w $m.f2
  frame $w -padx 1.5m -pady 1.5m
  ::tpad::AmpWidget checkbutton $w.stand -onv yes -offv no \
    -text [mc "error if not &standalone"] \
    -var ::tpad::xmlwf::$pn\::standalone
  ::tpad::AmpWidget checkbutton $w.ns -onv yes -offv no \
    -text [mc "describe &namespaces"] \
```

```
    -var ::tpad::xmlwf::$pn\::namespace
  pack $w.stand $w.ns -anchor w

  set w $m.but
  frame $w
  ::tpad::AmpWidget button $w.check -text [mc &Check] \
    -comm [namespace code "ProcessText $pn"]
  ::tpad::AmpWidget button $w.restore -text [mc "&Restore options"] \
    -comm [namespace code "InitSettings $pn; SetCheckbuttonState $pn"]
  ::tpad::AmpWidget button $w.close -text [mc Cl&ose] \
    -comm [namespace code "
      set ::tpad::$pn\::wf no
      focus $p.text
      after idle {destroy $m}
    "]
  pack $w.check $w.restore $w.close -side left -exp yes

  set w $m.log
  frame $w
  bind [::tpad::AmpWidget label $w.lab \
    -text [mc "&Last Error Message"]] \
    <<AltUnderlined>> [list ::tk::TabToWindow $w.txt]
  text $w.txt -width 40 -height 2 -relief raised -setgrid true \
    -state disabled  -font $::tpad::xmlwf::font -wrap none \
    -xscroll "$w.scroll set"
  scrollbar $w.scroll -orient horizontal -comm "$w.txt xview" \
    -width $::tpad::scrollwidth
  pack $w.lab $w.txt $w.scroll -fill x

  SetCheckbuttonState $pn
  grid $m.f -row 0 -column 0 -pady 2m -padx 2m
  grid $m.f2 -row 0 -column 1 -pady 2m -padx 2m
  grid $m.log -row 1 -column 0 -columnspan 2 -sticky ew -pady 2m \
    -padx 2m
  grid $m.but -row 2 -column 0 -columnspan 2 -sticky ew -pady 2m \
    -padx 2m

  bind $m <Alt-Key> [list ::tpad::AltKeyInDialog $m %A]
  bind $m <Key-Escape> [list ::tk::ButtonInvoke $m.but.close]

  ::tpad::PlaceWindow $pn $m

  return
```

```
}
```

Appendice B

Pacchettamento

Il *packaging* o *pacchettamento* di un programma consiste nel preparare un unico file di archivio, il cui formato dipende strettamente dal sistema operativo utilizzato, che faciliti all'utente l'installazione di software di terze parti.

I pacchetti possono contenere programmi in formato *binario*, ovvero già compilati per una particolare architettura hardware e quindi pronti all'uso, oppure *sorgenti*, che in molti casi si possono compilare per diverse architetture in modo del tutto o quasi automatico.

Generalmente un *pacchetto binario* contiene non solo tutti i file necessari per il corretto funzionamento del programma, ma anche alcune delle istruzioni di macchina necessarie per l'installazione automatica dello stesso sulla macchina dell'utente, i meta-dati che descrivono l'applicazione e la documentazione che spiega come usare il programma ai nuovi utenti.

Di solito un pacchetto binario non è portabile tra un sistema operativo e l'altro e spesso nemmeno tra versioni differenti dello stesso sistema operativo (ad es. diverse distribuzioni di Linux), nonostante il programma stesso sia portabile (è questo il caso della libreria tpad). Purtroppo non tutti i sistemi seguono perfettamente gli standard e gli standard stessi non coprono ogni minimo dettaglio necessario per garantire la piena portabilità. Il risultato è che gli sviluppatori di un programma non possono provvedere a creare il pacchetto per ogni possibile sistema in cui vi sono utenti interessati a quel programma, perché non hanno sufficiente interesse o risorse necessarie per farlo.

Soprattutto nel caso di software open-source il problema può essere risolto grazie al fatto che, essendo disponibile il sorgente, un programma può essere facilmente adattato a funzionare in un altro sistema anche da una persona diversa da coloro che hanno contribuito al suo sviluppo.

Infatti in inglese si usano i termini *packager* e *developer* per distinguere questi due ruoli. Il developer fornisce per lo meno delle istruzioni generiche su come fare per compilare e installare un programma. I packager seguono queste indicazioni adattandole alla specifica piattaforma in cui operano, risolvendo una volta per tutte eventuali problemi che si verificano durante l'installazione, in modo che non se ne debba preoccupare l'utente finale, che spesso non sarebbe in grado di farlo.

I *pacchetti sorgenti* invece sono utilizzati dagli stessi packager o da utenti più avanzati che desiderano avere un maggior controllo nelle fasi di configurazione e compilazione del software, allo scopo di decidere quali funzionalità includere e ottimizzazioni applicare. Alla fine da un pacchetto sorgente si genera un pacchetto binario configurato nel modo voluto e specifico della particolare piattaforma su cui si opera. Questo risulta utile soprattutto se si deve installare lo stesso programma su più macchine identiche, in quanto non si deve ripetere la compilazione su ciascuna macchina.

In alcuni sistemi, come FreeBSD e certe versioni di Linux, il sistema di packaging gestisce solo pacchetti binari e per la compilazione dei programmi a partire dal sorgente viene fornita un'altra tecnologia complementare, che generalmente va sotto il nome di *collezione delle port*, *port collection* o semplicemente *port*. Una port è una collezione di script, patch e altri file di dati che automatizzano il processo di compilazione di una applicazione dal codice sorgente. Una port è analoga ad un pacchetto sorgente; la differenza principale è che mentre il pacchetto sorgente contiene anche il codice del programma la port no. Generalmente il codice viene scaricato dal sito ufficiale del programma o da uno dei suoi mirror nel momento in cui si utilizza la port stessa. Questo consente di distribuire le port e i loro aggiornamenti indipendentemente dal codice su cui operano, realizzando un notevole risparmio di banda trasmissiva sui server centrali.

La funzione principale di un sistema di packaging è installare programmi da varie fonti, come i dischi fissi o rimovibili o tramite la rete, assicurando che nessuna coppia di programmi abbia mai due file in comune. In pratica però oltre la semplice installazione ci sono molte altre funzionalità che sono desiderabili in un software per la gestione dei pacchetti. Segue un breve elenco delle più comuni:

- disinstallazione automatica, con possibilità di preservare la configurazione del software, utile nel caso si decida poi di reinstallarlo
- funzionalità di sicurezza, tra cui la verifica della firma digitale o della integrità del pacchetto da installare o anche già installato
- aggiornamento a nuove versioni dei programmi in modo automatico o semi-automatico
- ricerca nel database dei meta-dati che descrive i programmi disponibili o già installati, in base a vari criteri
- installazione automatica delle dipendenze di un programma, ovvero degli altri programmi o librerie necessari al suo funzionamento

Il caso di tpad è un po' particolare, perché, oltre al fatto che il sorgente è portabile, tpad è già eseguibile in formato sorgente. Perciò i concetti di pacchetto binario e sorgente per tpad di fatto coincidono.

La distribuzione di tpad comprende un pacchetto sorgente generico che può essere installato su qualsiasi piattaforma hardware e sistema operativo che supporti Tcl/Tk e poi alcuni pacchetti specifici per le piattaforme più comuni, che non richiedono alcuna configurazione iniziale e quindi sono molto semplici da installare, proprio come un pacchetto binario.

Bibliografia

[1] Free Software Foundation. *GNU General Public License, Version 2, 1991.* `http://www.fsf.org/copyleft/gpl.html`

[2] *The FreeBSD Project.* `http://www.FreeBSD.org`

[3] A. Tridgell, P. Mackerras. *The rsync algorithm*, Australian National University of Canberra, TR-CS-96-05, 1996.

[4] *Tpad web site.* `http://tclpad.sourceforge.net`

[5] J. K. Ousterhout. *Tcl: An Embeddable Command Language*, Proc. USENIX Winter Conference, pp. 133-146, 1990.

[6] J. K. Ousterhout. *Scripting: Higher Level Programming for the 21st Century*, IEEE Computer magazine, March 1998.

[7] The Unicode Consortium. *The Unicode Standard Worldwide Character Encoding, Version 2.0*, Addison-Wesley, 1996.

[8] J. K. Ousterhout. *An X11 Toolkit Based on the Tcl Language*, Proc. USENIX Winter Conference, pp. 105-115, 1991.

[9] S. J. Peralta. *Scripting Graphical Commands with Tcl/Tk Mini-HOWTO.* `http://www.tldp.org/HOWTO/Scripting-GUI-TclTk`

[10] *TclPro Compiler.* `http://www.tcl.tk/software/tclpro/compiler.html`

[11] D. Ragget. *Clean up your Web pages with HTML TIDY.* `http://www.w3.org/People/Raggett/tidy/`

[12] R. Johnson. *Tcl Style Guide*, Sun Microsystems, 1997.

[13] *XFree86 web site.* `http://www.xfree86.org`