Università degli Studi di Padova

FACOLTÀ DI INGEGNERIA
Corso di Laurea Triennale in Ingegneria Elettronica

Tesi di Laurea

PLUG-IN PER L'AMBIENTE DI SVILUPPO ECLIPSE

PLUG-IN FOR THE ECLIPSE IDE

Novembre 2011

Candidato
Michele Gasparini
Matricola 524266-IL

Relatore
Prof. Franco Bombi

Anno Accademico 2011-2012

Indice

INTRODUZIONE

Nell'ambiente tecnologico, in particolare nel settore della produzione di semiconduttori, spicca il nome di *Infineon Technologies*, azienda multi-nazionale con quartier generale a Monaco e uffici sparsi in tutto il mondo.

Fondata nel 1999, può contare al giorno d'oggi sulla forza lavoro di più di 25.000 ingegneri alle sue dipendenze che, dislocati nelle oltre 20 sedi di ricerca e sviluppo dagli Stati Uniti all'Europa (Italia compresa) e all'Asia, apportano quotidianamente i loro contributo per mantenere l'azienda nella prestigiosa posizione di mercato in cui si trova da anni.

Nell'ultimo lustro Infineon Technologies ha sempre chiuso i bilanci annuali con cifre di tutto rispetto, facendo segnalare una crescita dei ricavi continua e costante che ha raggiunto nello scorso 2010 il valore di 3.295 miliardi di euro [1] .

Nello stesso anno, la qualità e l'affidabilità dei prodotti marchiati Infineon è stata insignita con il *Highest Level Quality Award* da Toyota Hirose Plant, premio che inquadra la multi-nazionale tedesca come ''la prima casa fornitrice di semiconduttori" al mondo a ricevere tale riconoscimento.

Le principali divisioni su cui è fondata la struttura aziendale, *Automotive*, *Industrial & Market* e *Chip Card & Security*, focalizzano le loro risorse nell'affrontare tre sfide che riguardano la società moderna: *energia*, *efficienza* e *mobilità e sicurezza*, offrendo al mercato semiconduttori e sistemi di soluzioni realizzabili, innovativi e di levatura universalmente riconosciuta.

[1] ''Annual Report 2010". Infineon Technologies. Retrieved 20 February 2011.

I successi più sostanziali provengono sicuramente dal segmento automobilistico che rifornisce case del calibro di AUDI e Renault con prodotti quali sensori, microcontrollori, driver LED e semiconduttori di potenza, tutti contraddistinti dal marchio *Zero-Defect* che è diventato negli anni sinonimo di garanzia e affidabilità.
Ed per l'appunto nel settore Automotive che impiega le forze la sezione R&D di Infineon Technologies Italia con sede a Padova, nella quale si è svolto il periodo di tirocinio.

All'interno del distaccamento patavino, l'unico presente nella penisola, le attività lavorative sono affidate a tre diversi team di ricerca: *Standard*, *Body Power* e *Microcontrollers*. Quest'ultimo, in particolare, si occupa di collaudo software (software testing), ossia della realizzazione di codice atto a testare il corretto funzionamento di uno specifico programma, e della validazione e caratterizzazione di memorie Flash integrate. E proprio il gruppo di microcontrollori ha ospitato e supportato lo stage universitario del cui progetto si tratta in questo esposto.

Capitolo 1

ECLIPSE

«Eclipse è uno strumento universale, un ambiente di sviluppo aperto ed estensibile per tutto e per niente in particolare. Eclipse rappresenta una delle iniziative più eccitanti nate nel mondo dello sviluppo delle applicazioni molti anni orsono, e gode del considerevole supporto delle compagnie e delle organizzazioni più importanti del settore della tecnologia.»

Con queste parole viene presentato Eclipse in quello che è universalmente considerato il miglior libro a riguardo: ''Eclipse Plug-ins'' (E. Clayberg, D. Rubel, Addison-Wesley Ed.).

Eclipse, fin dalla sua nascita, è stato apprezzato in tutti gli ambiti della produzione software, dalla progettazione alla commercializzazione, passando per la ricerca, lo studio e l'implementazione.

Si tratta di un ambiente di sviluppo collaudato, affidabile e scalabile, sulla base del quale si possono velocemente progettare, sviluppare e distribuire prodotti di eccellente qualità sia per finalità personali/aziendali che commerciali.

Il software non è coperto da diritti d'autore, e ciò ne consente la piena libertà di redistribuzione in tutto il mondo. Viene rilasciato sotto i termini della *Eclipse Public Licence*, che segue i principi e le regole fondamentali del software open source.

Lo sviluppo di Eclipse è portato avanti da una comunità assai vasta di

persone, che comprende sia singoli programmatori che contribuiscono per passione, sia grandi compagnie informatiche quali HP, IBM ed Eriksson. Il tutto a formare un ecosistema vastissimo, che va ben oltre la comunità dell'open source, fino includere prodotti commerciali Eclipse-based, riviste, portali online e fornitori di servizi di telecomunicazioni.

La struttura sulla quale si basa il progetto Eclipse è quella dei plug-in: ogni idea è portata avanti in maniera aperta e trasparente, coordinata a livello globale dalla *Eclipse Foundation*, un'associazione no-profit che è anche responsabile dell'approvazione e della crescita meritocratica dei progetti. Per raggiungere questo obiettivo, la fondazione deve garantire una costante comunicazione tra i membri a tutti i livelli della comunità, attraverso meeting annuali e con un'intensa pianificazione e capillare distribuzione dei compiti.

A dimostrazione della perfetta coordinazione e dei precisi meccanismi di interazione tra le parti, ogni nuova versione di Eclipse viene rilasciata con cadenza regolare di un anno: come si è potuto verificare a partire dal 2004, ogni mese di giugno viene ne distribuita una nuova release. Piccola curiosità, dalla versione 3.2 del 30 giugno 2006 alla 3.5 del 2009, tutte le distribuzioni portano il nome di uno dei satelliti principali di Giove (Callisto, Europa, Ganymede, Galileo).

1.1 Cenni storici

Alla fine degli anni '90, IBM si trovò di fronte ad un'impresa piuttosto ardua da portare a termine: aveva infatti pianificato, ed in parte già intrapreso, una migrazione verso un unico ambiente di sviluppo basato su codice Java. La tecnologia fino a quel momento utilizzata era un mix di C/C++, Smalltalk e Java, e molti dei software più importanti su cui IBM basava la propria produttività erano scritti proprio in Smalltalk, un linguaggio splendido per la costruzione di strumenti sofisticati, del quale gli sviluppatori di IBM avevano profonda padronanza. Al di fuori dell'azienda newyorkese Smalltalk non poteva però che contare su pochi supporter a livello di sviluppo software, e stava man mano per essere soppiantato in favore del più versatile Java, compatibile con un maggior numero di piattaforme e infinitamente più adatto

allo creazione di applicazioni web-based, grazie alla considerevole quantità di API per esso disponibili.

Così IBM incaricò il suo gruppo di programmatori scelti (OTI, Object Technology Internationl group), già affermatosi per la produzione della famiglia di piattaforme VisualAge, di creare un ambiente di sviluppo integrato altamente estensibile basato su Java. Eclipse fu la testimonianza del loro successo. Il mondo di Java non aveva mai assistito a nulla di così potente e allo stesso tempo avvincente come Eclipse che arrivava così a rappresentare, insieme a Microsoft's .NET, l'ambiente di sviluppo più prestigioso in circolazione.

Il team di IBM era riuscito a programmare un IDE che non aveva precedenti in termini di potenza, flessibilità e estensibilità.

Eclipse fu presentato al pubblico per la prima volta nel 2001, rilasciato gratuitamente sotto licenza *Common Public License*. Nel novembre dello stesso anno venne istituita la ''eclipse.org Board of Stewards'', una organizzazione mondiale nata dalla parternship di importanti brand del settore tecnologico e informatico: IBM, MERANT, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft and Webgain. A distanza di neanche due anni la fondazione poteva contare già su oltre 80 membri a sostegno di un progetto sempre più diffuso e in costante crescita. Nel 2004 venne annunciata la riorganizzazione della società nell'attuale *Eclipse Foundation*, una corporazione senza scopo di lucro che avrebbe rilasciato il proprio IDE secondo le norme della più liberale e specifica *Eclipse Public License*.
Nello stesso periodo altre prestigiose case produttrici entrarono a far parte del progetto. Apportano tutt'ora un considerevole contributo marchi del calibro di HP, Intel e MontaVista Software.

1.2 L'ambiente di sviluppo

Eclipse è un ambiente di sviluppo integrato (IDE, Integrated Development Environment) multi-linguaggio. Nonostante la prima versione servisse solo a programmare in Java, i linguaggi e le sintassi supportate al giorno d'oggi sono molteplici: C/C++, Perl, Phyton, Ada, COBOL, PHP, e tanti altri.

La piattaforma fu ideata con il chiaro intento di essere profondamente estensibile tramite plug-in scritti in Java, lo stesso linguaggio in cui è scritto l'intero software. E sono proprio la modularità e la possibilità di ampliamento i principali punti di forza che hanno portato Eclipse alla sua attuale diffusione. Senza contare l'indiscussa stabilità, la distribuzione in forma gratuita, e l'enorme supporto garantito dalla comunità alle sue spalle.

La release utilizzata durante il periodo di tirocinio è la 3.6 (nome in codice «Helios»), versione aggiornata di quella già in uso dagli ingegneri di Infineon Technologies Italia, nella particolare configurazione dedicata agli sviluppatori (di plug-in) Java messa a disposizione dal sito ufficiale *http://eclipse.org*.

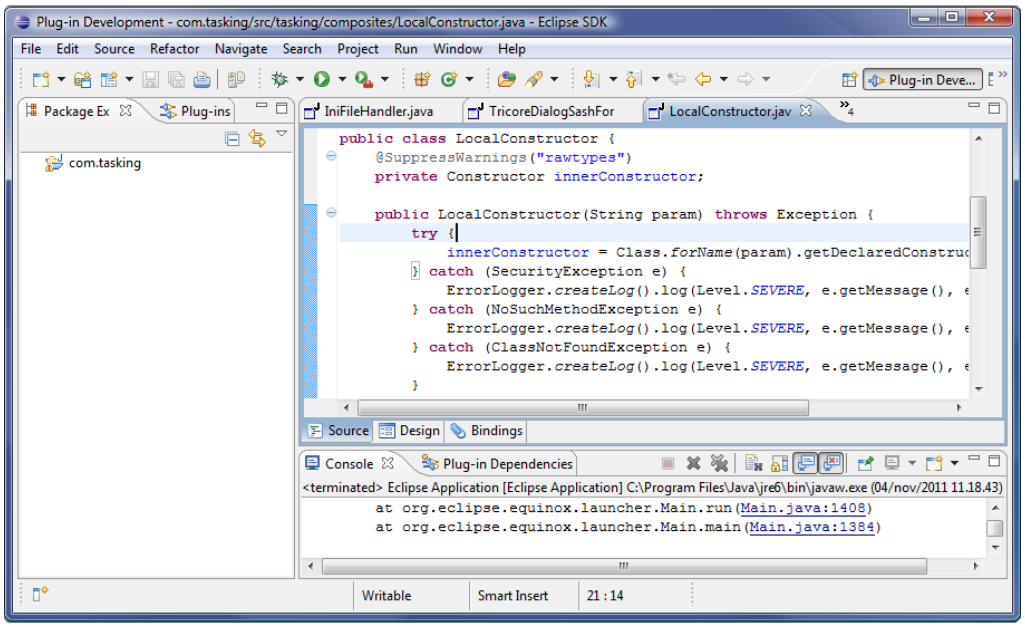

Figura 1.1: Interfaccia grafica dell'ambiente di sviluppo «Eclipse v3.6»

L'interfaccia grafica è piuttosto classica, decisamente ordinata e organizzata in settori di dimensioni variabili in proporzione all'importanza di utilizzo.
Il software mette a disposizione tutti gli strumenti indispensabili al programmatore: dall'immancabile editor di testo con riconoscimento della sintassi e navigazione multipla tramite schede, all'evoluto gestore di pacchetti appartenenti al progetto in fase di elaborazione. Non manca neanche il compilatore

integrato, che supporta già di serie una vasta gamma di linguaggi (in base alla versione scelta, comunque ampliabile attraverso plug-in) e la possibilità di eseguire o debuggare il codice in un ambiente di test virtuale.
Le funzionalità offerte da Eclipse sono davvero tantissime, ma se ancora non dovessero bastare sarà sufficiente collegarsi al sito *http://marketplace.eclipse.org/* per trovare migliaia di plug-in testati e pronti all'installazione.

1.3 Le librerie grafiche

Una delle caratteristiche che rende Eclipse incredibilmente potente e performante è la possibilità di poter contare su librerie grafiche ''ad hoc'', estremamente ottimizzate e complete.

Secondo gli standard di IBM, le piattaforme grafiche inizialmente distribuite da Sun Microsystem non si potevano considerare soddisfacenti sotto vari aspetti: AWT (Abstract Window Toolkit), il primo set di librerie deicate alla grafica rilasciato insieme alla JDK 1.0, metteva a disposizione dei programmatori uno strumento con cui creare GUIs (Graphical User Interfaces) per il loro software. Tuttavia, pur avendo diretto accesso alle componenti native del sistema operativo su cui il software veniva eseguito, come finestre, menù e pulsanti, AWT non era in grado di fornire un insieme di widget sufficientemente completo ed esaustivo.
Sun, resasi conto del disagio accusato dagli utenti, optò allora per intraprendere un nuovo percorso, abbandonando l'idea di sfruttare interfacce native del sistema operativo, per dedicarsi alla creazione di librerie proprietarie, portabili ed emulate. Sviluppò così le JFC (Java Foundation Classes), meglio conosciute sotto il nome di *Swing*. Pur offrendo un vasto set di oggetti grafici, Swing non fu mai apprezzato su larga scala per due principali motivi: dovendo ricreare da sé ogni singolo componente di ogni widget, senza potersi appoggiare sul sistema sottostante, risultava estremamente lento e ''pesante" in termini di risorse richieste, facendo sì che non potesse nemmeno essere eseguito all'interno della J2ME (Java 2 platform Micro Edition). In secondo luogo, l'elevata portabilità a cui aveva mirato Sun, ricorrendo all'esosa emulazione dei componenti grafici su JVM, portava inevitabilmente a un senso di

‘‘non appartenenza” delle interfacce grafiche al sistema operativo in uso, dal momento che con esso non condividevano praticamente alcun aspetto visuale.

Non in sintonia con la filosofia e le scelte di Sun Microsystem, IBM diede l'incarico al suo team di programmatori di creare un set di librerie grafiche altamente portabili, multi-piattaforma e che facessero uso delle risorse native condivise dall'OS, in modo da ridurre al massimo l'impatto sulle prestazioni. Il risultato fu un insieme di strumenti grafici scritto interamente in Java, eseguibile con impressionante rapidità, grazie all'efficace uso che esso fa della memoria del computer, e perfettamente adattabile all'interfaccia del sistema operativo su cui è eseguito. Prese il nome di *Standard Widget Toolkit* (SWT). Grazie a SWT, ogni oggetto visualizzato sembra essere stato pensato e realizzato appositamente per l'OS che lo esegue, e il portfolio di widget offerti è in grado di coprire ogni tipo di necessità.

Lo Standard Widget Toolkit si è velocemente affermato come la piattaforma su cui l'intera interfaccia utente di Eclipse si basa. Il progetto è tutt'ora portato avanti e sviluppato dalla Eclipse Foundation, i cui programmatori hanno più di recente rilasciato un nuovo set di librerie chiamato *JFace*. Si tratta di una sorta di involucro per oggetti di tipo SWT, che fornisce una ‘‘scorciatoia” per lo svolgimento di compiti che altrimenti richiederebbero un grosso dispendio in termini di tempo. JFace consente di racchiudere in un unico oggetto più widget di tipo SWT, permettendo il risparmio di numerose righe di codice.

Dunque, la potenza, la completezza e le performance delle librerie SWT, unite alla praticità e alla comodità del layer JFace più ad alto livello, hanno decretato il successo di Eclipse non solo come ambiente di sviluppo tuttofare, ma anche come ottimo strumento per la creazione di GUIs per software di ogni genere e circostanza.

Capitolo 2

IL PLUG-IN

In questo capitolo viene descritto il plug-in dal punto di vista progettuale e strutturale. Partendo dalle motivazioni che hanno spinto alla realizzazione di un simile progetto, si giungerà all'analisi della struttura interna del software, non prima però di aver avuto una panoramica generale sul plug-in e una accurata precisazione sulle scelte tecniche adottate durante il percorso.

2.1 Come nasce il progetto e qual'è il suo scopo

La realizzazione del plug-in per l'ambiente di sviluppo Eclipse fa parte di un'idea ben più ampia partorita dalla mente dell'ingegner Antonio Fin, insostituibile membro del team *Microcontrollers* di Infineon Technologies Italia, e di alcuni altri collaboratori del team stesso.

Il progetto generale nacque da un'esigenza primaria che gli sviluppatori si trovarono a dover soddisfare: avere a disposizione uno strumento che racchiudesse al suo interno una serie di funzioni offerte fino a quel momento da diversi software in nessun modo interfacciati tra loro.
Durante il loro quotidiano lavoro infatti, i programmatori del team Microcontrollers si trovano di continuo a dover scrivere, correggere e compilare codice c/c++ o Java, ricorrendo all'uso di programmi proprietari esterni per la selezione delle opzioni di compilazione. Uno dei software di utilizzo più diffuso per assolvere a tale necessità all'interno dell'azienda è il *Tasking Tri-*

Core, prodotto dalla australiana Altium e in grado di supportare la varie famiglie di prodotti di Infineon.
Va inoltre precisato che una volta scritto e compilato il sorgente, gli ingegneri hanno anche il compito di aggiornare la versione del software in fase di sviluppo, di debuggare il codice, di inviare un'email agli altri team per comunicare il rilascio di un aggiornamento o di una nuova release del programma, e molti altri step fondamentali al funzionamento di una catena produttiva ben organizzata e coordinata che porta ad un prodotto di eccelsa qualità universalmente riconosciuta.

Tutto ciò impone l'impiego di numerosi tool che porta inevitabilmente ad una notevole perdita di tempo. Ecco quindi nascere l'idea di avere a disposizione uno strumento comodo, pratico e versatile, una sorta di ''coltellino svizzero" da tenere a portata di mano per svolgere le operazioni più frequenti. La comodità di utilizzo di un tale software si sarebbe tradotta perciò in un cospicuo risparmio in termini di tempo, e mai come in un'azienda altamente coordinata a livello internazionale la tempestività è stata un fattore chiave.

2.2 Panoramica del progetto

La base di partenza, unica vera certezza al momento del lancio del progetto, era l'ambiente di sviluppo *Eclipse* che con la sua modularità e la sua possibilità di personalizzazione attraverso plug-in scritti in Java, aveva tutte le carte in regola per adempiere alle richieste dei progettisti Infineon.

Il plug-in in progetto doveva fornire in primo luogo una via d'accesso facilmente raggiungibile a tutte le più comuni opzioni per la creazione di *make* file e *linker* file (entrambi strettamente legati al compilatore associato), supportando i diversi tipi di compilatori utilizzati durante la fase di test del software (tra cui diverse versioni di Tasking, GCC, GNU, Hitex, ...).
Doveva inoltre consentire il *software versioning* di file e progetti in fase di elaborazione, nonché una fase di debug del codice, mettendo a disposizione sia la riga di comando, sia un'interfaccia grafica snella e confortevole da usare.
Ancora una volta, la scelta di Eclipse si è dimostrata vincente, dal momento

che Tigris.org rilascia gratuitamente tool opensource per il versionamento del software perfettamente compatibili e integrabili nell'IDE. Per completarne le funzionalità, il plug-in doveva supportare l'invio di *bug* e *request* mail, contenenti segnalazioni di bachi nel codice o richieste ad altri sviluppatori di apportare modifiche in determinate sezioni di programma. Per fare ciò, doveva essere fornito all'utente un semplice form grafico, con pochi campi di testo da riempire e alcuni menù selettivi.
La modularità e la scalabilità del progetto di un unico ambiente di sviluppo testware avrebbe permesso futuri ampliamenti e miglioramenti del plug-in, con l'eventuale aggiunta di componenti e funzioni ritenute utili al programmatore.

2.3 Scelte progettuali

Fin dai primi giorni di lavoro, durante la fase di studio di fattibilità del progetto, ci si è resi conto che gli obiettivi inizialmente prefissati erano troppo azzardati: in un periodo di tirocinio di ''sole" 250 ore, di cui una prima parte comunque spesa nello studio e nell'apprendimento delle conoscenze basilari sulle librerie grafiche di Java e nell'acquisizione di familiarità con l'ambiente Eclipse, era pressoché impensabile di riuscire a realizzare e sviluppare l'intera struttura software preventivata.
Dal momento che alla fine dello stage lavorativo doveva essere consegnato un prodotto completo e funzionante, pronto per l'utilizzo e perfettamente in grado di svolgere le funzioni implementate, si è pensato di concentrare gli sforzi solo su alcune delle parti che componevano il progetto originale, quelle ritenute di impiego più immediato e frequente. L'idea era quella di mantenere il software il più modulare possibile, garantendo la possibilità di integrazioni e aggiornamenti successivi.

Si è stabilita allora la linea guida da seguire per l'intera durata della fase di sviluppo: il plug-in avrebbe dovuto fornire un'interfaccia grafica chiara, facilmente accessibile e richiamabile all'interno di Eclipse, che permettesse di selezionare e modificare tutte le opzioni di compilazione tipiche di Tasking TriCore, dando come output un file facilmente leggibile e interpretabile sup-

portato da diversi compilatori.
La sicurezza e la stabilità del software non dovevano passare in secondo piano: un file di back-up, copia esatta di quello di uscita, avrebbe garantito un buon livello di sicurezza dei dati, mentre un sistema di log degli errori avrebbe tenuto traccia di eventuali problemi in fase di esecuzione o crash del programma. Il plug-in avrebbe inoltre dovuto permettere la possibilità di versionamento del software, compito, come già detto, pienamente assolto dal tool distribuito sotto licenza gratuita e opensource da Tigris.org.

2.3.1 Interfaccia grafica

Nella valutazione dell'interfaccia grafica la scelta ricadde su quella che sembrava essere l'opzione più logica: realizzare delle finestre ''Tasking look and feel'', ossia che riproducessero l'aspetto e le funzioni del programma già sfruttato in Infineon (Figura 2.1), così da rendere il più indolore possibile il passaggio alla nuova piattaforma.

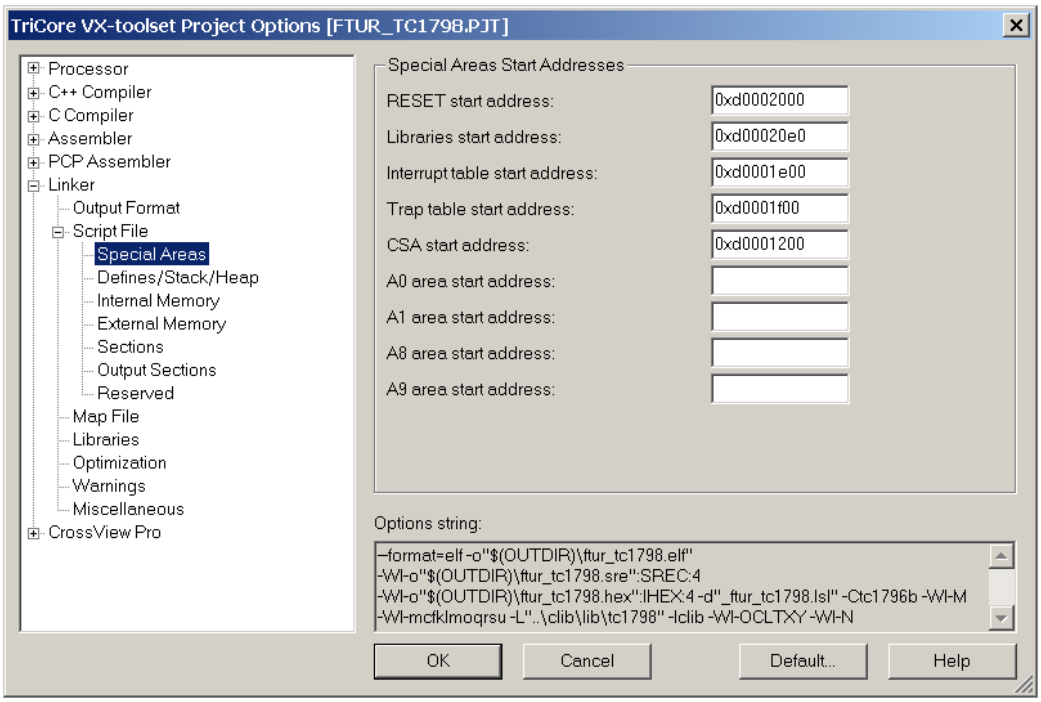

Figura 2.1: Interfaccia grafica del software «Tasking TriCore»

Si optò per un'unica finestra grafica, divisa verticalmente in due riquadri principali di dimensioni differenti: in quello di sinistra prese posto una lista composta di elementi selezionabili, che sostituisce la struttura ad albero espandibile presente nel software di casa Altium, mentre la parte destra (più ampia delle precedente) avrebbe ospitato le varie pagine di opzioni visualizzate in seguito al click su uno dei componenti della lista stessa.
Si decise anche di implementare nel plug-in solo le opzioni offerte da Tasking realmente utilizzate dai programmatori, tralasciando tutte le funzionalità superflue, in modo da alleggerire il software sia in termini di occupazione di memoria, sia in termini di pulizia grafica. Inoltre ci si impose di migliorare alcuni dettagli che nel Tasking TriCore potevano sembrare obsoleti: così, ad esempio, i campi di testo a singola riga a scorrimento orizzontale vennero sostituiti da campi testuali multi-riga con scrolling verticale.
Per realizzare il tutto si fece ragguardevole impiego di oggetti appartenenti alle librerie grafiche SWT e JFace messe a disposizione da Eclipse.

2.3.2 Output del software

Dopo l'interfaccia grafica di input, l'attenzione è stata spostata sull'output del programma. I file fino a quel momento utilizzati come dati di ingresso per i compilatori o per la traduzione attraverso appositi script erano quelli generati da Tasking TriCore. Si trattava di oggetti di tipo .pjt, una sorta di file di testo contenente al suo interno un riepilogo di tutte le opzioni selezionate dall'utente attraverso l'interfaccia del software. Tali file risultavano però molto difficili da interpretare alla lettura, ponendo non pochi ostacoli durante le operazioni di ricerca e sezionamento del loro contenuto, al fine di eliminare o aggiungere alcune righe.

Come uscita del nuovo plug-in si pensò allora di avere qualcosa di più facilmente gestibile, chiaro da scorrere in lettura e ordinato in sezioni al suo interno. Inoltre doveva essere ampiamente supportato dai diversi sistemi operativi e dai più comuni linguaggi di programmazione.
Questi requisiti restringevano sostanzialmente la cerchia dei possibili candi-

dati a due sole tipologie di file: il .ini e il .xml. Il secondo era certamente la soluzione più moderna e versatile, ma anche più complessa da maneggiare. Lo standard adottato fu invece l'*ini* che, seppur datato e recentemente rimpiazzato in alcuni campi dal xml, possedeva le caratteristiche necessarie al progetto, senza rendere troppo complicato il suo utilizzo. La sua longevità, vista positivamente in termini di consolidamento e sviluppo, garantiva il giusto equilibrio tra supporto e funzionalità. Inoltre, il set di API (Application Programming Interface) *ini4j* metteva a disposizione una completissima serie di strumenti con cui lavorare il formato ini in ambito di programmazione Java senza dover rimpiangere le più avanzate peculiarità di xml.

Il file di tipo .ini è un particolare file di testo che gode di una semplice struttura basata sulla divisione in *sezioni*. Ogni sezione è a sua volta composta di *parametri*, ossia classiche coppie "chiave-valore" tipiche dei dizionari. La basilare struttura permette una chiara lettura del contenuto del file, oltre che la ricerca di sezioni o chiavi al suo interno e la possibilità di enumerare le voci presenti in esso. Il pacchetto ini4j mette a disposizione un'ampia gamma di comode funzioni per operare su tali file, tra cui non può ovviamente mancare l'aggiunta, la modifica ed l'eliminazione di sezioni o singoli parametri.

```
; Sample Ini File

[SECTION_NAME]
parameter_1 = value_1
parameter_2 = value_2

[ASSEMBLER_PREPROCESSING]
Tasking preprocessor = true
No preprocessor = false
Define user macros = false
Include this file before source = true
Include '.def' file = false
Path = ./Eclipse/plugins

[LINKER_SPECIAL_AREAS]
RESET start address = 0xD4000000
Libraries start address = 0xD4001000
Interrupt table start address = none
Trap table start address = none
CSA start address = 0xD0002000
```

Figura 2.2: Esempio struttura del file ini

La semplicità del formato ini fa si che esso sia agevolmente interpretabile e traducibile attraverso script (verosimilmente in linguaggio *perl*, il più utilizzato in azienda per assolvere a compiti di questa entità) così da poter adattare i dati in uscita dal plug-in ai diversi compilatori da supportare.

2.3.3 Backup automatico

Per salvaguardare il lavoro dell'utente e per garantirne in ogni momento la sicurezza dei dati, si è pensato di dotare il software di un sistema automatico

di backup: esso provvede autonomamente a creare una copia identica del file ini (in formato .bkp) ad ogni lancio del plug-in. Ciò comporta un duplice vantaggio: da un lato si ha sempre a disposizione un duplicato del file contenente tutte le opzioni impostate nella sessione precedente, scongiurando il rischio di perdere tutto il lavoro svolto in caso di corruzione del file ini per crash di sistema o errori in esecuzione del programma. Dall'altro lato, essendo la copia di backup creata all'avvio del plug-in, si ha sempre a portata di mano il contenuto del file ini prima che vengano applicate su di esso le modifiche apportate nella sessione di lavoro in corso.
Questa particolarità è di fondamentale importanza per il corretto funzionamento del plug-in nel momento in cui si desideri scartare gli ultimi cambiamenti effettuati piuttosto che salvarli: è proprio dal file .bkp infatti che il software preleva le informazioni per ripristinare il file .ini. Si approfondirà questo delicato passaggio nel capitolo ''Manuale d'uso".

2.3.4 Sistema di log degli errori

Nonostante si sia cercato in tutti i modi di rendere stabile il codice del plug-in, sia mantenendo la massima linearità strutturale (seguendo le regole basilari del *clean code*) sia facendo uso di tutti i costrutti ''try-catch'' per catturare eventuali eccezioni a runtime, vi è sempre la possibilità che si verifichi un errore o addirittura un blocco inaspettato nell'esecuzione del software. E poiché l'evenienza si presenta anche con i programmi più blasonati commercializzati da prestigiose case informatiche, non abbiamo di certo la presunzione di scartarla a priori nel nostro modesto contesto!

Si è deciso allora di inserire nel plug-in un modulo che tenesse traccia errori, eccezioni, o quant'altro, riportando il tutto all'interno di un file di log in formato txt. Così, nell'eventualità di comportamenti inattesi e/o inappropriati da parte del software, l'utente sarà sempre in grado di ottenere utili informazioni sulla natura del malfunzionamento, nella speranza di riuscire a risalirne alla causa per correggere ipotetici bachi o possibili mancanze nell'utilizzo.
Per raggiungere lo scopo si è fatto uso della classe *Logger* del package ja-

va.util.logging, che mette a disposizione uno strumento efficiente e facilmente configurabile, e che crea un file di testo molto dettagliato contenente ogni tipo di informazione relativa all'errore verificatosi, ad esempio ora e data, classe, metodo e riga che ha lanciato l'eccezione, oltre che una breve descrizione del tipo di anomalia.

2.4 Come si compone il software

Dal punto di vista strutturale, il plug-in è scritto interamente in Java. Il progetto è composto di tre package (Figura 2.3) che racchiudono tutte le classi necessarie al funzionamento del software. Elencandoli in ordine di complessità crescente, si trova:

- package *constants*;
- package *handlers*;
- package *composites*;

Il primo pacchetto si limita a contenere un'unica classe, *Constants*, al cui interno sono presenti i percorsi delle directory in cui sono situati i file letti o generati dal programma, oltre che alcune costanti numeriche utilizzate dal plug-in. Tale classe ha semplicemente lo scopo di agevolare eventuali interventi di modifica a variabili presenti in molte parti del codice, e che (se non fossero così rappresentate sottoforma di costanti in un unico file) andrebbero ricercate e corrette una ad una tra migliaia di righe.
La composizione degli altri due packages, invece, è ben più articolata e merita di essere trattata in maniera più approfondita e dettagliata.

2.4.1 Package *handlers*

Il package *handlers* include sei file, ognuno rappresentante una classe Java. Come lascia intuire il suo nome, la funzione principale di questo pacchetto è di *gestire* e controllare i file creati o letti dal programma. Osservando più da vicino il contenuto si incontrano le seguenti classi:

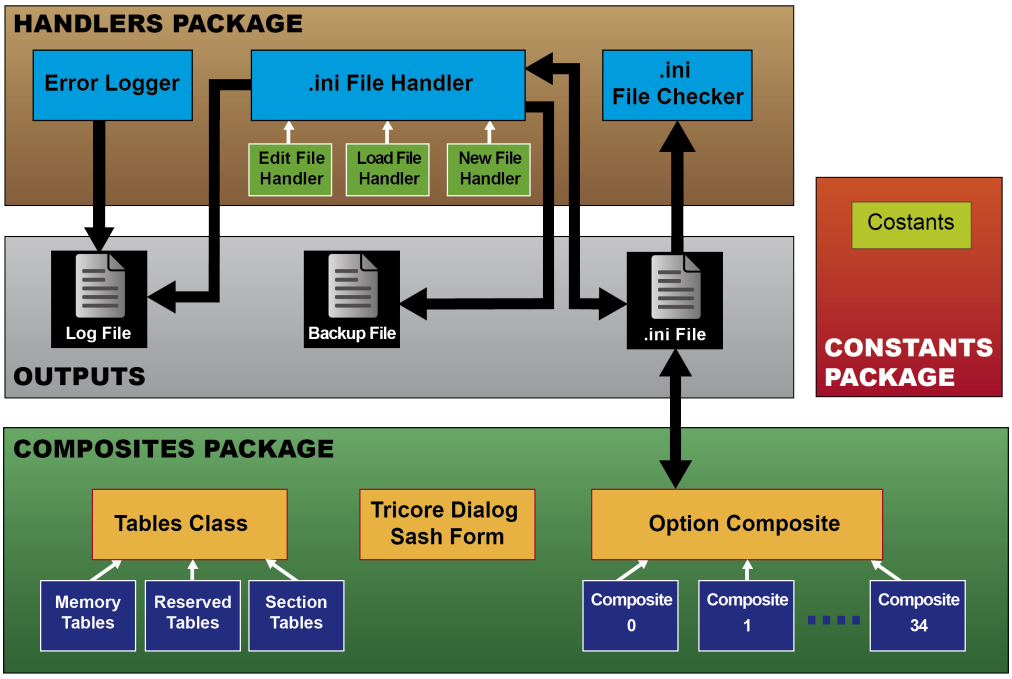

Figura 2.3: Struttura interna del software. Si noti in particolare come le principali classi interagiscano con i file di input/output.

- **ErrorLogger**: si occupa di creare il file di log degli errori, assegnandogli una formattazione, e provvedendo a fornire un'oggetto che consenta la scrittura sul file log.txt delle eventuali eccezioni generate dal software.

- **IniFileChecker**: ha come unico scopo la verifica dell'integrità del file ini in fase di caricamento, sia esso richiamato tramite comando di *load* o di *edit* (si veda il capitolo "Manuale d'uso" per i dettagli su tali comandi). Attraverso un apposito oggetto in grado di analizzare strutture di tipo ini, questa classe si accerta che all'interno del file caricato sia presente il corretto numero di sezioni, e che ognuna di esse non sia corrotta, ma che presenti cioè il giusto numero di coppie chiave-valore.

- **IniFileHandler**: è la classe più importante del package, dalla quale ereditano le sottoclassi specifiche *NewFileHandler*, *LoadFileHandler* e *EditFileHandler*. Svolge un ruolo basilare per il funzionamento del

software che ha inizio con la visualizzazione sullo schermo della finestra principale del plug-in e delle finestre di dialogo attraverso le quali l'utente è tenuto a indicare il percorso del file ini da utilizzare per il progetto da elaborare. Una volta ottenuto il path assoluto del file, la classe IniFileHandler lancia la routine di controllo dell'effettiva presenza del file, di accertamento di integrità dello stesso, e del suo ripristino in caso di danneggiamento rilevato. È anche responsabile della comparsa a display di vari pop-up, ad esempio quello di richiesta di inserimento del nome del file ini da creare (in caso ci si accinga a iniziare un nuovo progetto), o di sovrascrittura di un file già presente nel percorso specificato, o ancora di conferma di ripristino di un file danneggiato (per fare ciò ci si avvale di un *Template.ini* salvato nella cartella di installazione del plug-in, contenente tutte le sezioni e i relativi parametri impostati a valore di default).
Ultimi, ma non per importanza, i compiti di generare e tenere aggiornato il file di backup in una specifica directory il cui percorso è indicato in una delle costanti della classe Constants, e di creare (o all'occorrenza di pulire) il file di log degli errori ad ogni esecuzione del software.

2.4.2 Package *composites*

L'ultimo package, nominato *composites*, è il più vasto e complesso del progetto. Ha come funzione primaria quella di creare tutta la parte grafica del software, dalla finestra principale in cui è eseguito il plug-in ai più piccoli riquadri di dialogo visualizzati durante l'utilizzo del programma stesso. Come detto, il numero di classi che compone il package è ben più ampio rispetto a quello dei precedenti pacchetti, perciò si procederà all'analisi dettagliata solamente di quelle di rilevanza maggiore:

- **TricoreDialogSashForm**: è indubbiamente la classe ''maestra" del package. Deve il suo nome alla funzione principale che svolge: questa classe è infatti preposta alla creazione della struttura grafica del "*dialog*" in cui viene eseguito plug-in, ossia una finestra separata dall'ambiente di sviluppo Eclipse, che comprende i pulsanti ''OK'', ''Cancel" e

"Default..." oltre che la *sashform* [1] che ne occupa gran parte dell'area. L'aspetto di tale finestra ricorda in molti particolari l'interfaccia di Tasking TriCore, dove la parte sinistra della sashform è riempita da una lista di elementi clickabili che, una volta selezionati, comportano la visualizzazione del corrispondente pannello di opzioni nella parte di destra.

- **OptionComposite**: è la classe delegata alla realizzazione del riquadro di opzioni che viene raffigurato volta per volta all'interno della sashform. Il compito di caratterizzare il contenuto di ciascuno dei 35 diversi pannelli disponibili è affidato invece alle sottoclassi *Compisite_ n* (dove n rappresenta appunto un indice numerico compreso tra 0 e 34) che estendono la superclasse OptionComposite. Quest'ultima però, oltre che ad una mansione puramente grafica, è anche addetta alla scrittura e alla lettura dei dati all'interno del file ini: ogniqualvolta, infatti, che si seleziona un diverso elemento della lista di sinistra, e che viene quindi visualizzato il corrispondente riquadro di opzioni, specifici metodi della classe OptionComposite vanno a leggere la relativa sezione del file ini, convertendo i dati codificati nei singoli parametri per rappresentarli graficamente nella finestra del plug-in. Risulta in questo modo facile intuire come ad ogni componente della lista selezionabile (e quindi ad ogni pannello di opzioni) corrisponda una particolare sezione del file ini, le cui singole righe altro non sono che la codifica testuale di ciascun campo editabile, pulsante, o menù presente nella correlativa pagina grafica.
  Allo stesso modo, al momento del passaggio da un elemento della lista a un altro, i dati presenti nel riquadro corrente vengono scritti nella rispettiva sezione del file ini, pronti per essere convertiti e ripristinati all'occorrenza.

[1] sashform: si tratta di uno spazio di forma rettangolare, posto all'interno di una finestra grafica, diviso in due o più parti al cui interno possono essere alloggiati vari widget. La particolarità dell'oggetto sashform consiste nel poter ridimensionare in maniera dinamica le sezioni che lo compongono in base alla grandezza della finestra che lo contiene.

- **TablesClass**: al fine di rispecchiare al massimo tutte le funzionalità del software Tasking TriCore, vi è stata necessità di creare delle tabelle editabili, in cui le varie celle potessero contenere campi modificabili di diverso genere: combo menù (altrimenti detti menù a tendina), campi testuali e caselle di spunta.
  Poiché le librerie standard di Java non provvedono a fornire un simile oggetto già confezionato e pronto all'uso, per soddisfare le richieste è stata implementata la classe TablesClass, a sua volta estesa dalle sottoclassi *MemoryTable*, *ReservedTable* e *SectionsTable* che costituiscono i diversi tipi di tabella impiegati nel programma. Essa mette a disposizione un oggetto grafico di tipo tabella modificabile in cui è possibile scegliere, a seconda del proprio bisogno, il numero e il tipo di righe e colonne, specificando il genere di cella di cui devono essere composte. Vi è inoltre la possibilità di aggiungere o rimuovere dinamicamente specifiche righe nel punto in cui lo si desideri.

2.4.3 Il file *plugin.xml*

Sebbene non appartenga a nessuno dei pacchetti precedentemente analizzati, e nonostante non sia scritto in Java, il file *plugin.xml* costituisce una parte fondamentale del software. In esso sono infatti contenute tutte le informazioni relative ai link per lanciare l'esecuzione del plug-in. All'interno del file sono quindi indicate le modalità con cui richiamare il programma, siano esse tramite pulsante sulla barra delle applicazioni di Eclipse, oppure tramite menù contestuale situato nella barra dei menù dell'ambiente di sviluppo.
Il file plugin.xml contiene inoltre indicazioni su quelle che devono essere le icone raffigurate dai pulsanti e sulle azioni che Eclipse deve intraprendere una volta che questi sono premuti.

Capitolo 3

MANUALE D'USO

Il presente capitolo ha lo scopo di illustrare il plug-in dal punto di vista funzionale, mostrando le caratteristiche che esso mette a disposizione dell'utente e descrivendo le modalità in cui l'utilizzatore debba farne uso.

Quanto segue parte dall'assunto che il plug-in sia già stato correttamente installato all'interno dell'ambiente di sviluppo Eclipse. Nell'eventualità che tale prerequisito non sia soddisfatto, è possibile fare riferimento alla breve guida di installazione dell'appendice A.

3.1 Come si presenta il software

Una volta installato il plug-in e avviato Eclipse, si nota la presenza di una nuova voce nella barra dei menù, *Tasking Options* (Figura 3.1) che, se estesa, mostra tre differenti diciture: *Load .ini File*, *New .ini File* e *Edit .ini File*. Ciascuna di esse serve a lanciare l'esecuzione del software in modalità diverse.

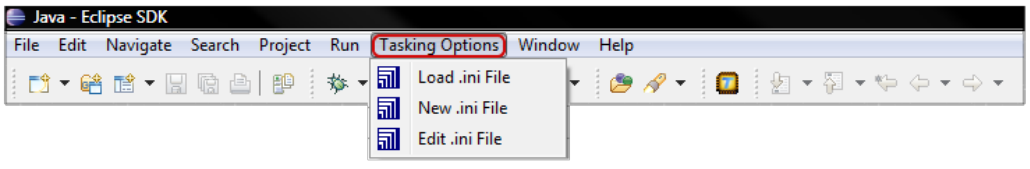

Figura 3.1: Barra dei menù di Eclipse dopo l'intsllazione del plug-in.

A sua volta, anche la barra degli strumenti presenta un nuovo bottone

 il quale altro non è che un accesso rapido alla funzione Edit .ini File.

La finestra principale in cui è eseguito il plug-in (Figura 3.2) pone innanzi ad una grafica semplice e ordinata, i cui stile e aspetto risultano perfettamente integrati con il sistema operativo che la sta visualizzando grazie all'adozione delle librerie SWT e JFace.

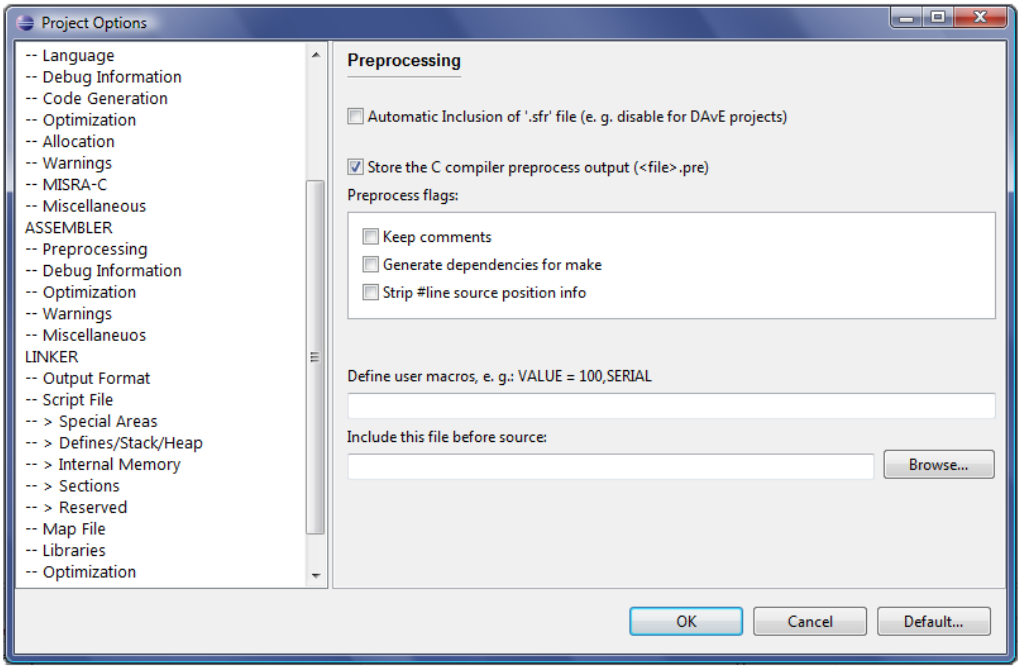

Figura 3.2: Finestra principale in cui è eseguito il plug-in.

Divisa verticalmente in due riquadri, mostra nella parte di sinistra una lista scorrevole dall'alto al basso, nella quale ogni elemento rappresenta una diversa sezione del software su cui si andranno a impostare le specifiche opzioni di compilazione. Ognuna delle 35 sezioni (alcune delle quali evidenziate con caratteri maiuscoli ad indicare i principali settori di lavoro) viene rappresentata nel pannello destro della finestra, che ne occupa gran parte della superficie, in seguito al click del mouse sulla corrispondente voce della lista. Ed è proprio all'interno dell'area di opzioni che l'utente ha la possibilità di intervenire sulle varie impostazioni agendo su campi di testo, tabelle, caselle di spunta e menù selettivi "a tendina".

Tutte le modifiche apportate attraverso l'interfaccia grafica verranno poi codificate e salvate all'interno del file ini.

3.2 Interagire con il plug-in

Come già accennato, l'esecuzione del plug-in può essere avviata in tre diverse modalità attraverso l'apposito menù *Tasking Options* presente nella barra di Eclipse:

- LOAD ini FILE: tramite questa opzione, l'utente è in grado di caricare nell'interfaccia del software il contenuto di un file ini già esistente. In una finestra pop-up separata viene quindi chiesto all'utilizzatore di indicare il file da aprire sfogliando graficamente le cartelle dell'hard-disk. Un apposito filtro verifica che l'estensione del file selezionato sia corretta, evitando l'apertura di oggetti che non sarebbero altrimenti interpretabili dal programma. Fatto ciò, l'utente è invitato a inserire un nome per il file che verrà generato: tutti i cambiamenti apportati da questo momento, infatti, non saranno salvati sul file importato, bensì su una copia di esso creata all'interno del *workspace* di Eclipse (cartella predefinita nella quale l'IDE archivia i file relativi al progetto) e rinominata con il nome appena digitato. In questo modo il file originale resterà integro e sarà sempre riutilizzabile in futuro.
  Nell'eventualità che il nome scelto appartenga ad un file già esistente nella directory di default, un pop-up presenterà la possibilità di modificarlo oppure di sovrascrivere il vecchio file con il nuovo (Figura 3.3).

- NEW ini FILE: consente la creazione, per default all'interno del workspace di Eclipse, di un nuovo file ini su cui cominciare a lavorare. In questo caso, l'utente è tenuto solamente a fornire un nome da assegnare al file, che sarà poi inizializzato con una serie di valori standard prelevati automaticamente da un modello situato nella cartella di installazione del plug-in.
  Ancora una volta, se nel workspace esiste già un file con la stessa etichetta, verrà chiesto se sovrascriverlo oppure no.

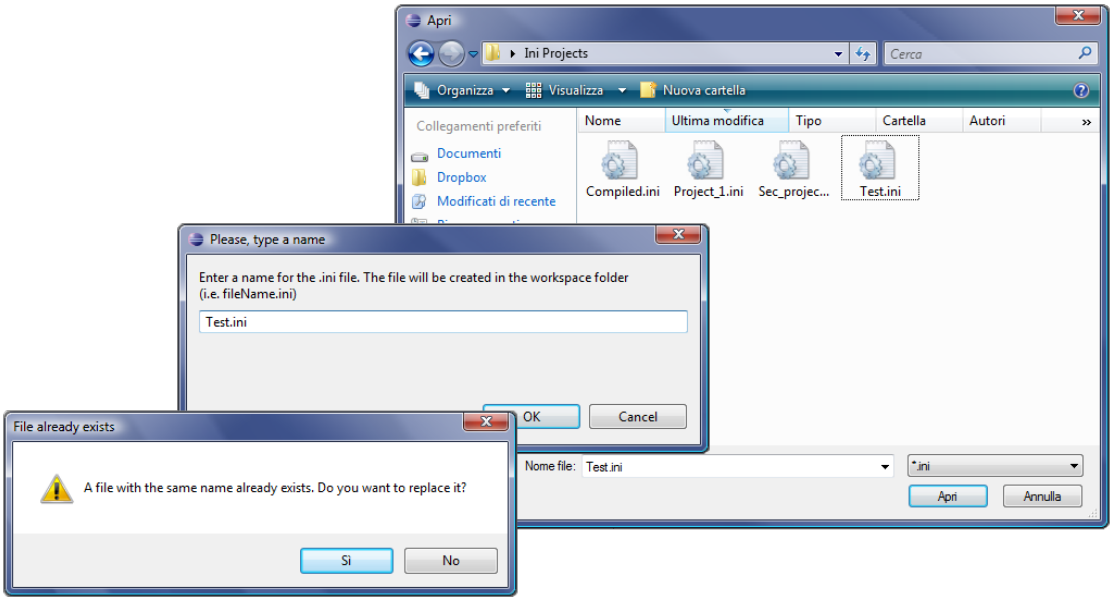

Figura 3.3: Richiesta sovrascrittura di un file esistente.

- EDIT ini FILE: è l'unica tra le modalità di lancio del plug-in ad offrire un accesso veloce tramite un tasto collocato sulla toolbar di Eclipse, in quanto si è constatato essere l'opzione più frequente durante l'utilizzo del software Tasking. Scegliendo di editare un file ini, si ha la possibilità di importare all'interno dell'interfaccia grafica un progetto già in fase di sviluppo. Diversamente da quanto accade però selezionando la modalità LOAD, in questo caso tutte le modifiche apportate al file ini aperto vengono salvate direttamente su di esso, e non in un nuovo file separato.

Una volta scelta la modalità di utilizzo più appropriata ed effettuate le poche operazioni preliminari, comparirà sullo schermo la finestra principale del plug-in con i vari parametri impostati secondo necessità.

Durante questo passaggio, il software provvede automaticamente a creare all'interno della cartella di installazione del plug-in un file denominato *log.txt* (nel caso esista già, si limita a cancellarne tutto il contenuto per prepararlo ad una nuova scrittura) nel quale verranno elencati tutti gli errori e le eccezioni prodotti da eventuali malfunzionamenti del programma.

Parallelamente viene anche generata una copia bit-a-bit del file ini appena aperto, il cui contenuto è "congelato" fino al termine della sessione di lavoro. Tale backup, che di default è allocato nella directory workspace, offre maggiore sicurezza e affidabilità al plug-in in caso di crash inaspettati del plug-in o di errori durante la sua esecuzione. Ma, oltre ad una funzione puramente cautelativa, il file di backup svolge un ruolo cardine nelle operazioni di salvataggio e scarto delle informazioni elaborate, come verrà analizzato nell'immediato seguito.

3.2.1 Salvataggio e annullamento delle modifiche

Come si nota fin dal primo momento, l'interfaccia del plug-in è sprovvista di un tasto *Salva*. Questo perché il software agisce in maniera totalmente automatizzata. Cosa significa: nell'istante in cui l'utente abbia concluso le operazioni in una determinata scheda di opzioni sarà sufficiente passare ad un'altra voce della lista per far si che le modifiche fin lì apportate vengano memorizzate all'interno del file ini. Un click su un diverso elemento della lista è tutto quel che serve per far in modo che il programma prenda in carico tutti i parametri della pagina appena modificata e li codifichi nella corrispondente sezione del file ini.

Il tasto *OK* svolge la medesima funzione: nel caso non si intenda andare ad effettuare altre variazioni, ma sia solo necessario salvare il lavoro svolto e e terminare l'esecuzione del plug-in, la pressione del bottone OK comporta la chiusura della finestra grafica e l'archiviazione dei dati nel file ini, senza il bisogno di cambiare pannello di opzioni.

Si supponga ora di voler annullare ogni cambiamento effettuato durante la corrente sessione di lavoro, ma di non ricordare ogni singola operazione svolta fino al dato momento. Per il raggiungimento di tale scopo viene in aiuto il tasto *Cancel*. La sua pressione fa sì che l'esecuzione del plug-in venga terminata istantaneamente e che il contenuto del file ini vengano ripristinato alla condizione desiderata, ossia alla versione non modificata dall'ultima elaborazione.

Tale operazione è possibile grazie alla presenza del file *Backup.bkp* che il plug-in crea o aggiorna ad ogni suo avvio. La peculiarità del file di backup consiste proprio nel fatto di non essere ''toccato" fintanto che il software non viene lanciato, mettendo sempre a disposizione dell'utente (o del programma) una copia del file ini allo stato in cui si trovava prima delle modifiche subite in una determinata fase di lavoro. Gli stadi di generazione e recupero dei file ini e backup sono illustrati nella Figura 3.4.

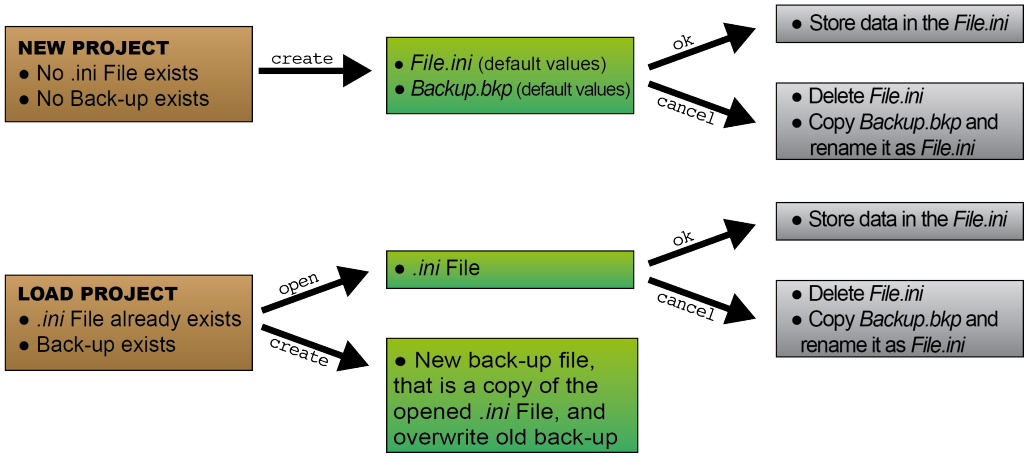

Figura 3.4: Generazione e recupero dei file ini e backup.

Come si evince dall'immagine, il processo di scarto delle modifiche si basa su due semplici passaggi (totalmente automatizzati): il file *nome_file.ini* appena elaborato (e di cui non interessano le variazioni) viene eliminato, mentre il file backup.bkp viene duplicato e la nuova copia rinominata in nome_file.ini.

L'ultimo pulsante a presenziare nella barra inferiore del dialog principale è *Default...*; esso consente con un solo click di reimpostare ai valori predefiniti tutti i campi editabili di tutti i pannelli di opzioni del software. Ancora una volta, i parametri di riferimento vengono recuperati dal file Template.ini situato nella directory di installazione del plug-in (.../Eclipse/plugins/com.tasking.plugin.v1.0.0.first).

3.3 Modificare il file di uscita

Il file ini in cui il plug-in codifica e archivia le impostazioni manipolate graficamente dall'utente può essere a sua volta ispezionato e modificato "a mano". Trattandosi infatti di un comune file di testo (con alcune proprietà specifiche aggiuntive), può essere comodamente aperto per mezzo di un qualsiasi editor testuale e corretto secondo le proprie esigenze. Va ricordato infatti che, per natura stessa dello standard ini, il contenuto del file di output è facilmente leggibile e decifrabile. Le sezioni al suo interno portano lo stesso nome di quelle dell'ambiente grafico (in modo tale da agevolarne la ricerca), così come i parametri e i valori ad essi associati sono chiaramente riconoscibili una volta acquisita una certa familiarità con l'interfaccia del plug-in.
Dopo aver salvato il file e lanciato l'esecuzione del plug-in tramite GUI, tutti i cambiamenti apportati da riga di testo verranno rielaborati e visualizzati correttamente sulla finestra grafica.

Questa particolarità consente un utilizzo *bi-direzionale* del software, permettendo di intervenire sulle opzioni di compilazione messe a disposizione dal plug-in sia a livello grafico, sia a livello testuale, qual'ora fosse necessario un intervento più mirato direttamente nel file di uscita. Quest'ultima possibilità può rivelarsi utile, per esempio, nel caso in cui si sia già provveduto alla chiusura dell'IDE Eclipse e ci si trovi a dover modificare un piccolo dettaglio nel file ini: non si avrà dunque l'esigenza di riavviare l'ambiente di sviluppo e attendere che venga caricato l'intero sistema, ma sarà sufficiente editarne l'output per ottenere il risultato voluto.

Capitolo 4

ASPETTI IRRISOLTI

Nonostante i notevoli sforzi effettuati per consegnare un prodotto completo e funzionante entro il limite temporale imposto dal periodo di tirocinio, e sottolineando come il plug-in sia effettivamente utilizzabile in ogni suo aspetto pratico, alcuni particolari operativi restano ancora da migliorare o integrare.

L'ostacolo più ostico da scavalcare, e tutt'ora non superato, riguarda le tabelle modificabili. Nelle tre sezioni del plug-in nominate rispettivamente *Internal Memory*, *Sections* e *Reserved* sono presenti altrettante tabelle grafiche su cui l'utente può intervenire in varie modalità: è possibile editare campi testuali, selezionare elementi all'interno di menù a tendina, aggiungere righe in qualsiasi punto lo si desideri.
Un'oggetto così sofisticato, ovviamente, non viene fornito già "pacchettizzato" e pronto all'uso da nessun set di librerie grafiche. Lo si è perciò costruito ricorrendo all'utilizzo di oggetti e metodi appartenenti alla classe *TableViewer* e ad altre classi minori messe a disposizione dalle librerie JFace.

Un'operazione che sarebbe molto utile poter eseguire, ma che non si è riusciti a implementare, è la rimozione di una riga (sia essa intermedia o finale) della tabella. Ciò è dovuto al complicato sistema di refresh che le tabelle editabili impiegano per visualizzare in tempo reale le modifiche apportate, oltre che alla loro complessa struttura che fa si che risulti problematico spostare intere righe in alto una volta che ne viene eliminata un'altra.
Tale mancanza ha ripercussioni non soltanto a livello di utilizzo delle stesse

tabelle grafiche, ma anche sul metodo di controllo di integrità del file ini.

La classe IniFileChecker, per assolvere al compito di verifica di un file aperto o caricato, esegue prima un conteggio del numero di sezioni presenti all'interno del file, e successivamente un ulteriore conteggio del numero di parametri elencati in ognuna di esse. L'efficienza del controllo si basa sul fatto che il numero di sezioni così come quello dei parametri è stabilito in fase progettuale e non variabile (ad eccezione appunto delle tre sezioni relative alle tabelle). Risulta pertanto chiaro che, nel caso di rimozione (ma anche di inserimento) di una riga rispetto alla dotazione standard delle tabelle, il test sull'integrità del file ini fallirebbe impedendo l'esecuzione del software. Per ovviare a tale inconveniente si è provveduto a disabilitare il sistema di verifica per le sole sezioni corrispondenti alle tabelle, lasciandolo attivo per tutte le altre trentatré.
E' sicuramente in programma, tuttavia, per i prossimi step progettuali, l'elaborazione di metodi che consentano l'eliminazione di righe dalle tabelle e il controllo delle rispettive sezioni nel file ini.

Un altro aspetto del plug-in non perfezionato, per cause imputabili alla sola mancanza di tempo residuo, è il sistema di log degli errori. Esso sfrutta oggetti di tipo *Logger* messi a disposizione dal package standard java.util.logging che si sono rivelati un valido punto di partenza per futuri sviluppi di oggetti di log "ad hoc".
Nella personale fase di testing condotta in azienda durante il periodo di stage, è stata messa in evidenza, infatti, una certa difficoltà da parte del pacchetto offerto dalle librerie Java nella gestione di molteplici errori occorsi nella singola sessione di utilizzo del plug-in. Va comunque precisato che l'anomalia descritta non ha portato ad alcun malfunzionamento del software, nè ad occasioni di instabilità dello stesso, ma si è rispecchiata solamente nella creazione di diversi file di log temporanei (a fianco del vero e proprio log.txt) non opportunamente cancellati al termine dell'esecuzione del plug-in.
Nulla di grave, dunque, dal punto di vista funzionale; si tratta piuttosto di un dettaglio che andrebbe forse curato con maggior scrupolosità e più dedizione in termini temporali.

Capitolo 5

CONCLUSIONI

Seppur con i dovuti aggiustamenti e dopo un netto ridimensionamento, il progetto è stato realizzato rispettando le richieste e le specifiche imposte. Le scelte tecniche effettuate durante la fase di studio (si vedano, tra le altre, l'adozione dello standard ini e l'introduzione del sistema di backup) si sono rivelate assai funzionali e sono state apprezzate in tutto l'ambiente aziendale.

Il plug-in, dopo la breve fase di test, si è dimostrato all'altezza dei propositi iniziali, con doti di semplicità ed efficienza nella media, riuscendo anche a rispettare i vincoli (non scritti) si leggerezza e bassa occupazione di memoria.

Restano ancora da risolvere alcune, poche, questioni relative all'implementazione del software; va tuttavia precisato che tali imperfezioni non influiscono in maniera rilevante sulle possibilità di utilizzo del plug-in.

Sarebbe forse stata consigliabile una più approfondita sessione di test, resa probabilmente troppo sbrigativa dalla breve durata del periodo di tirocinio. Il plug-in resta comunque un prodotto finito e funzionante, degno di una prima release.

Appendice A

INSTALLAZIONE DEL PLUG-IN

Il plug-in viene distribuito sottoforma di archivio *Zip*, il quale contiene al suo interno tre diversi files:

- un file in formato *Jar* che costituisce il software vero e proprio;
- il modello (*Template.ini*) nel quale sono riportati tutti i valori di default da assegnare a un file ini standard;
- la *release note* che fornisce informazioni riguardanti la versione del software, lo stadio di sviluppo e una breve descrizione delle funzionalità.

Procedere all'installazione è un'operazione tanto veloce quanto elementare: è sufficiente infatti estrarre il contenuto del pacchetto Zip all'interno della cartella *plugins* presente nella directory di installazione di Eclipse (es. .../Eclipse/plugins). Chiudere la finestra principale dell'IDE (se in esecuzione) e riavviare il programma. Il plug-in è pronto per l'utilizzo.

Bibliografia

[1] E. Clayberg, D. Rubel ‘‘Eclipse Plug-ins’’, 3rd Ed., Addison-Wesley 2008.

[2] M. Scarpino, S. Holder, S. Ng, L. Mihalkovic, ‘‘SWT/JFace in Action’’, Manning 2005.

[3] http://www.eclipse.org/org/

[4] http://www.vogella.de/eclipse.html

[5] http://www.infineon.com/cms/en/corporate/company/index.html

[6] Infineon Technologies, ‘‘Annual Report 2010’’, Retrieved 20 February 2011

[7] http://en.wikipedia.org/