A Francesco e Anna, miei dolci genitori:
a voi cui tutto devo.

Silenzio assordante… Eco smarrita…
Frastuono di preghiera inconsolata sussurrata nella notte.

Introduzione

Nemo propheta in patria

Prefazione

Probabilmente gran parte dei lettori di questo libro conosceranno svariati trattati relativi alla programmazione Object Oriented, magari in Java, e avranno partecipato alla costruzione di sistemi informatici più o meno complessi. Altrettanto probabilmente, una buona percentuale avrà constatato che non sempre il modo di produrre sistemi software segue un approccio, per così dire, ingegneristico — basato su un processo formale — e che spesso il linguaggio di modellazione unificato (UML, *Unified Modeling Language*) recita un ruolo troppo marginale. Il problema è che, sebbene lo UML sia un linguaggio — magari di modellazione ma pur sempre un linguaggio — non sempre è chiaro come, quando e perché utilizzarlo. Ebbene, il presente libro si prefigge l'obiettivo di fornire al lettore tutte le informazioni necessarie per utilizzare lo UML nel contesto dei processi di sviluppo del software, utilizzando un approccio operativo che però non trascuri l'imprescindibile teoria.

Il libro è stato scritto riducendo al minimo i prerequisiti, e conferendo particolare importanza agli esempi, specialmente quelli tratti da progetti reali, nella convinzione che un particolare legame con la realtà quotidiana dell'ambiente lavorativo sia il mezzo più idoneo per focalizzare rapidamente i concetti esposti… nonché per procurare all'autore vitto e alloggio a spese dello Stato. Ciò nondimeno si è cercato di non trascurare il pubblico più esigente, inserendo nel libro diversi paragrafi di livello tecnico superiore. Si tratta tuttavia di sezioni dedicate a lettori che desiderano l'approfondimento, ben circoscritte e assolutamente opzionali, quindi non indispensabili alla comprensione di quanto esposto. Il testo pertanto si presta a essere fruito secondo diversi livelli di difficoltà, in base a necessità e obiettivi del lettore.

Questo libro conferisce particolare enfasi a due grandi tematiche portanti: analisi dei requisiti utente e analisi e disegno del sistema. Quindi gli strumenti dello UML analizzati più approfonditamente sono sia quelli utilizzati durante le relative fasi del processo di sviluppo del software sia quelli utilizzati più frequentemente nel ciclo di vita del software: anche in questo contesto — come rimarcato dallo stesso James Rumbaugh — vale la famosa equazione dell'80/20: "l'80% della modellazione di un sistema, tipicamente, necessita del 20% dei concetti dello UML".

Ma quali sono i motivi che hanno spinto a scrivere un altro libro dedicato allo UML? Le risposte potrebbero essere due e molto semplici:

- perché è in lingua italiana (o almeno in un suo surrogato);
- perché i libri dei Tres Amigos (James Rumbaugh, Ivar Jacobson, Grady Booch), pur essendo uno straordinario ausilio — guai a non averne una copia, anche se mai aperta: si rischierebbe un giudizio di scarsa professionalità —, risentono forse del limite di essere poco accessibili a un pubblico non espertissimo di progettazione Object Oriented.

L'idea di fondo di questo libro è fornire un diverso livello di astrazione: i libri dei Tres Amigos offrono un livello molto elevato — quasi filosofico — mentre il presente vuole essere decisamente vicino alla pratica nel tentativo di fornire un supporto concreto a tutti coloro che utilizzano, o vorrebbero utilizzare, lo UML quotidianamente per costruire sistemi che funzionano. A tal fine sono stati forniti numerosi suggerimenti provenienti dall'esperienza pratica, sono stati messi in luce "tranelli" in cui è possibile cadere e si è fatto ricorso a moltissimi esempi che, sebbene finiscano per rendere i capitoli decisamente corposi, sono comunque sezioni opzionali sebbene ricche di utili riflessioni.

La scrittura del libro è iniziata quasi congiuntamente alla pubblicazione della versione 1.2 dello UML e abbraccia un periodo di due anni di lavoro "matto e disperatissimo". I contenuti sono costantemente revisionati e aggiornati sia al fine di renderlo via via conforme alle nuove specifiche, sia per integrare esempi sempre più interessanti attinti dall'esperienza lavorativa. Nella sua versione attuale, la variante dello UML di riferimento è la 1.4, sebbene l'autore stia controllando da vicino i lavori relativi alla versione 2.

Tra le righe del libro fa capolino l'(auto)ironia dell'autore, coadiuvata all'occorrenza da quella dei collaboratori: ciò sia al fine di rendere l'argomento meno gravoso, sia per evidenziare con spirito grottesco le distorsioni riscontrate nel modo di produrre il software in molte realtà organizzative. Troppo spesso i vari manager sono sfrenatamente concentrati a consegnare i sistemi senza curare minimamente la crescita professionale del proprio personale e quindi, in ultima analisi, la qualità dei sistemi prodotti. Non è il software che rifiuta l'ingegnerizzazione, ma i tecnici che rimangono intimoriti dalla progettazione: troppo spesso si ha la sensazione che la produzione del software sia ancora concepita come un'arte piuttosto che come una scienza.

A chi è rivolto

Questo libro prevede un pubblico di potenziali lettori piuttosto vasto: tutti coloro che hanno a che fare con progetti basati sul paradigma Object Oriented (o Component Based). In particolare, potrebbe risultare vantaggioso per figure professionali che vanno dai tester ai programmatori, dagli architetti ai capi progetto, dai business analyst a, perché no, persone con ruoli a carattere più commerciale.

Gli **architetti** costituiscono sicuramente il pubblico ideale: loro, in effetti, dovrebbero essere in grado di utilizzare proficuamente il linguaggio durante tutta la fase di sviluppo del software, dall'analisi dei requisiti, al disegno del sistema, ai test di accettazione.

I vantaggi che potrebbero ricavarne i **programmatori** scaturiscono essenzialmente dalla capacità di leggere i diagrammi forniti dagli architetti, per tradurli in codice e infine aggiornarli per far sì che il disegno finale corrisponda alla reale implementazione del sistema. Non è peccato mortale se l'implementazione varia, entro certi limiti, dal modello di disegno sia perché non è opportuno scendere eccessivamente in dettaglio nel disegno del sistema, sia perché non è sempre possibile prevedere tutto fin dall'inizio, inclusi malfunzionamenti di software fornito da terze parti. Nel mondo ideale la codifica dovrebbe essere immediata e senza troppe variazioni: nella realtà non è sempre così. Variazioni in corso d'opera sono normali fintantoché non stravolgano il disegno stesso. Sempre i programmatori, durante la codifica del disegno, dovrebbero avere bene in mente i casi d'uso oggetto dell'implementazione: essi descrivono sia la sequenza di azioni da svolgere nel caso in cui tutto funzioni correttamente, sia le misure da attuare per gestire eventuali anomalie che potrebbero verificarsi.

Per ciò che concerne i **capi progetto**, in molti casi essi potrebbero aumentare la qualità del proprio lavoro utilizzando i diagrammi dei casi d'uso (*use cases diagram*) e più in generale la relativa vista (*use cases view*, vista dei casi d'uso), al fine di coadiuvare il team nella "cattura" dei requisiti utente nel miglior modo possibile, monitorare adeguatamente lo stato di avanzamento del progetto, riuscire ad avere una più intima conoscenza di cosa si sta costruendo. Il cliente alla fine verifica che il sistema realizzi correttamente quanto sancito nella *use case view* presente nel documento di analisi dei requisiti.

I **team leader**, tra le altre attività, potrebbero trarre enorme beneficio dall'utilizzo dello UML per assolvere alle tipiche funzioni di coordinamento, supporto e addestramento del team e per coadiuvare i capi progetti nell'analisi della tempificazione al fine di renderla quanto più reale possibile… La tempificazione non è esclusivo esercizio di fantasia: tempificare con successo non significa dare i numeri con Microsoft Project.

I **tester** dovrebbero essere in grado di validare, o, se si preferisce, di individuare malfunzionamenti eseguendo i famosi test a scatola nera (*black box tests*) per verificare che il sistema effettivamente aderisca alle specifiche catturate sempre dalla *use case view*.

Per terminare, i **business analyst**, nel loro lavoro di analisi dell'organizzazione del cliente e della sua attività, potrebbero utilizzare i diagrammi dello UML per realizzare il famoso documento dei requisiti utente, modellando una prima versione del modello dei casi d'uso

e del diagramma a oggetti del dominio. Ciò apporterebbe due validi risultati: semplificazione del lavoro dei tecnici responsabili delle fasi seguenti e accrescimento del livello di professionalità e della qualità della documentazione prodotta.

Consigli per la lettura

Visto il grande impegno profuso nel redigere il presente libro, verrebbe voglia di consigliarne una lettura basata sul seguente algoritmo:

```
Iterator pagineLibro = libroUML.iterator();
while ( pagineLibro.hasNext() ) {
      utente.read( pagineLibro.next() )
}
```

Chissà che ciò possa essere realmente utile e possa fornire validi spunti soprattutto a coloro che non hanno molta esperienza dello UML e dei processi di sviluppo.

Ciò nonostante, la struttura del libro è stata fortemente caratterizzata dalla consapevolezza di quanto limitata e preziosa sia la risorsa tempo. Si è cercato quindi di organizzare i contenuti in modo da favorire tutti coloro che sono desiderosi di arrivare rapidamente al nocciolo degli argomenti senza investire troppo tempo in tematiche ritenute meno interessanti. In particolare vi sono sezioni di elevata complessità dedicate ai lettori più esigenti, che però possono essere assolutamente ignorate senza per questo precludersi la possibilità di comprendere quanto riportato negli altri paragrafi.

Altre parti invece sono dedicate a considerazioni relative al processo di sviluppo: di nuovo, i lettori che volessero concentrare l'attenzione unicamente sullo UML possono semplicemente sorvolare tali paragrafi.

Infine, vi sono interi paragrafi dedicati allo stile da utilizzare per disegnare i vari diagrammi. Sebbene non siano di importanza fondamentale, questi paragrafi non sono comunque originati da un vezzo estetico. Si tratta di una serie regole che permettono di rendere i diagrammi più eleganti, armoniosi, semplici, in poche parole più accattivanti. Ciò è molto utile perché rende i diagrammi più facilmente comprensibili, memorizzabili, concorre a ridurre le barriere psicologiche che spesso si innalzano nella mente dei fruitori dei diagrammi, ecc.

I primi due capitoli (*UML: che cosa è, che cosa non è* e *UML: struttura, organizzazione, utilizzo*), considerata la genericità degli argomenti affrontati, possono risultare dispersivi e a tratti tediosi. Sebbene non siano di importanza fondamentale, risultano utili per chiarire fin da subito cosa sia lo UML e il suo rapporto con le tecnologie attigue. A partire dal capitolo relativo ai casi d'uso, tutto diviene più operativo e quindi scorrevole: si entra nel vivo dello UML!

Infine è presente un numero decisamente elevato di esempi attinti dal mondo del lavoro; tutti coloro certi di aver capito possono placidamente concentrarsi solo su alcuni di essi ignorando i restanti.

Struttura

Questo libro è stato strutturato in capitoli autonomi, in ognuno dei quali viene esaminato approfonditamente uno specifico argomento, cercando di limitare al massimo i rimandi agli altri capitoli: non è quindi necessario rincorrere gli argomenti per tutto il testo. Si è tentato di ricostruire una sorta d'unità d'azione: Milan Kundera non approverebbe. Ciò, se da un lato offre evidenti vantaggi, dall'altro genera la necessità di introdurre inevitabili ridondanze…

Capitolo 1 – UML: che cosa è, che cosa non è

Obiettivo del primo capitolo è presentare lo UML inquadrandolo nel suo contesto, chiarendo fin da subito il rapporto con le altre tecniche/metodologie che lo circondano.

Il capitolo si apre con una breve panoramica storica dello Unified Modeling Language e delle motivazioni che hanno spinto i Tres Amigos ad affrontare lo straordinario progetto di un linguaggio di modellazione unificato. Si prosegue con una descrizione di cosa sia e cosa non sia lo Unified Modeling Language, della sua architettura e di quali siano gli obiettivi.

Successivamente viene affrontato il tema dei più diffusi processi di sviluppo del software e, in particolare, The Unified Software Development Process, Use Case Driven Object Modeling with UML, il Rational Unified Process (RUP), l'eXtreme Programming (XP) e l'Agile Modeling (AM).

Capitolo 2 – UML: struttura, organizzazione, utilizzo

Il secondo capitolo è interamente dedicato a una panoramica dello UML: vengono presentate le viste, i diagrammi e i meccanismi di estensione forniti dallo UML per aumentarne semantica e sintassi. Sono inoltre introdotti i profili, ossia collezioni di meccanismi di estensione predefiniti da utilizzarsi durante la modellazione di sistemi che utilizzano architetture standard quali ad esempio CORBA ed EJB (*Appendice C*) o per compiti specifici come la modellazione dell'area business.

L'obiettivo è fornire un'idea, quanto più esauriente possibile, di cosa sia il linguaggio, senza aver la pretesa che il lettore comprenda fin da subito tutti i concetti introdotti, per i quali sono presenti appositi capitoli di dettaglio. Importante è iniziare a comprendere che lo UML fornisce tutta una serie di "utensili", ciascuno dei quali risulta appropriato a scopi ben precisi e del tutto inadeguato ad altri: una vera e propria cassetta degli attrezzi.

La descrizione dei vari manufatti che si prestano a essere realizzati per mezzo dello UML avviene nel contesto di un generico processo di riferimento.

Capitolo 3 – Introduzione agli Use Case

Nel terzo capitolo si focalizza l'attenzione sulla teoria degli use case e sul relativo impiego rispettando rigorosamente le direttive dello UML. Nel capitolo seguente invece, ci

si discosterà sensibilmente dalle direttive standard, al fine di illustrare un utilizzo avanzato e maggiormente operativo dei casi d'uso.

Dopo una breve disquisizione su cosa si intenda con i termini "requisiti utente", si procede con l'introduzione della use cases view (vista dei casi d'uso), entrando finalmente nel vivo dello UML.

Della vista dei casi d'uso sono descritte le due componenti: statica e dinamica. Per ciò che riguarda la proiezione statica, sono descritti in dettaglio i diagrammi dei casi d'uso correlati da elementi e relazioni. Per ciò che concerne la proiezione dinamica (ossia gli scenari), viene illustrata la versione embrionale basata sui flussi degli eventi e sui diagrammi dello UML atti a modellare comportamenti dinamici (diagrammi di sequenza, collaborazione, attività, ecc.).

Capitolo 4 – Modellazione avanzata degli Use Case

Nel quarto capitolo si chiarisce definitivamente quanti e quali siano i modelli dei casi d'uso presenti in un processo di sviluppo del software formale e viene illustrata una tecnica di utilizzo operativo per la cattura e la documentazione degli *scenario* dei casi d'uso. Si tratta di un metodo basato sull'utilizzo di opportuni template che a volte si discosta sensibilmente dalle direttive standard. Modellare sistemi reali mette in luce una serie di problemi e lacune di cui si tratta raramente nei libri e nelle specifiche ufficiali dello UML.

Si tratta di un capitolo di stampo decisamente pratico corredato da un notevole numero di esempi di casi d'uso pseudoreali. Sebbene non sia indispensabile esaminarli attentamente tutti, ognuno di essi è stato selezionato in quanto fornisce una serie di spunti molto interessanti.

Capitolo 5 – Completamento dell'analisi dei requisiti

Il quinto capitolo è imperniato sue due tematiche portanti: la presentazione di una tecnica dimostratasi particolarmente valida nell'arduo compito dell'analisi dei requisiti utente e l'illustrazione dei manufatti (*artifact*) che completano il modello dell'analisi dei requisiti.

Il problema che tenta di risolvere la tecnica di analisi dei requisiti esposta è trovare una piattaforma di dialogo comune e costruttiva tra personale tecnico e clienti, in altre parole trovare un efficace raccordo tra i due diversi linguaggi tecnici: quello dell'area business e quello informatico. A tal fine viene sfruttato il formalismo degli activity diagram (diagramma delle attività).

Il modello dei casi d'uso, sebbene costituisca un artifact di importanza centrale nel contesto dei processi di sviluppo del software, da solo non è assolutamente in grado di fornire sufficienti informazioni per il pieno svolgimento delle restanti fasi del ciclo di vita del sistema. La completa realizzazione della vista dei requisiti del sistema prevede la pro-

duzione di ulteriori manufatti complementari. In particolare, nel capitolo sono presentati artifact di notevole importanza, come il dettaglio delle interfacce, i test case, il documento dei requisiti non funzionali e le famosissime business rules.

Capitolo 6 – Object Oriented in un chicco di grano

Il sesto capitolo è dedicato alla presentazione dei princìpi fondamentali dell'Object Oriented. L'obiettivo è introdurre le nozioni utilizzate nei capitoli successivi, senza aver la presunzione di trattare in maniera approfondita un argomento così importante e vasto.

Il capitolo non poteva che iniziare con la definizione formale di oggetto e classe e delle relative relazioni, prosegue con l'illustrazione degli elementi essenziali di cui è composto un oggetto (identità, stato e interfaccia) e quindi presenta le tre leggi fondamentali dell'Object Oriented (ereditarietà, incapsulamento e polimorfismo). In questa sede si coglie l'occasione per riportare alcuni errori tipicamente commessi da tecnici alle prime armi. Questa prima sezione termina con l'introduzione di due princìpi fondamentale del disegno dei sistemi, note con i nomi di "massima coesione" e "minimo accoppiamento". La parte conclusiva del capitolo è dedicata al formalismo dell'*Abstract Data Type* (tipo di dato astratto) e al *Design By Contract* (disegno per contratto).

Capitolo 7 – Gli oggetti: una questione di classe

Il settimo capitolo è focalizzato sull'illustrazione del formalismo dei diagrammi delle classi, probabilmente uno dei più noti e affascinanti tra quelli definiti dallo UML. Nei processi di sviluppo del software, questo formalismo si presta a rappresentare formalmente molteplici artifact presenti in diversi stadi del processo. Nelle primissime fasi lo si adotta per rappresentare i modelli del dominio e di business, nello stadio di analisi è utilizzato per dar luogo all'omonimo modello, nella fase di disegno è impiegato per progettare e documentare l'organizzazione in codice del sistema e così via.

Dopo aver illustrato come rappresentare in UML i concetti basilari di classi, interfacce, ecc., si passa all'illustrazione dettagliata delle diverse tipologie di relazioni. La modellazione di un sistema non richiede esclusivamente l'identificazione delle classi di cui è composto, ma anche dei legami che le interconnettono.

La sezione successiva del capitolo è dedicata all'illustrazione del formalismo dei diagrammi degli oggetti (*object diagram*). Questi rappresentano una variante dei diagrammi delle classi, o meglio ne sono istanze e ne condividono gran parte della notazione. Rappresentano la famosa istantanea del sistema eseguita in un preciso istante di tempo di un'ipotetica esecuzione.

Il capitolo si conclude con l'illustrazione di una serie di esempi completi e con l'immancabile sezione dedicata allo stile grafico.

Capitolo 8 – Le classi nei processi

Il capitolo ottavo è dedicato all'illustrazione dell'utilizzo del formalismo dei diagrammi delle classi nel contesto di un processo di sviluppo del software. In particolare si presentano le diverse versioni di modelli a "oggetti", le loro proprietà, un nutrito insieme di esempi e consigli, eventuali "tranelli" in cui è possibile cadere, il rapporto di ciascuno di essi con i restanti, ecc. Si tratta di uno di quei capitoli che dovrebbe fornire risposte esaurienti agli interrogativi più frequentemente posti all'autore del libro.

Il primo modello ad oggetti presentato è quello relativo al dominio del problema (*Domain Object Model*). In particolare si presenta un processo pratico atto a semplificare l'individuazione delle classi appartenenti al modello e a inserirle in un contesto strutturato. Del processo viene fornito un esempio di applicazione, nel quale sono evidenziati errori ricorrenti e vengono dati utili consigli.

Poiché è sempre meno frequente realizzare sistemi partendo dal nulla, mentre è ormai prassi reingegnerizzare sistemi esistenti, si presenta una tecnica atta a realizzare prime versioni del modello a oggetti del dominio partendo dall'analisi di una base di dati del sistema.

Il modello ad oggetti presentato successivamente è quello relativo all'area business (*Business Object Model*). Si tratta di una versione molto simile a quello del dominio, in cui compaiono elementi aggiuntivi (*work unit*, unità di lavoro e *worker*, lavoratori). Pertanto si evidenziano le similitudini e le differenze con quello del dominio e si dà quindi una serie di consigli relativi a quale usare, quando, e così via. Teoricamente, in un processo di sviluppo andrebbero realizzati entrambi; nella pratica, per via di una serie di problemi relativi a tempo, budget, personale, ecc. è normale trascurarne uno per concentrasi sull'altro.

Il passo successivo consiste nel presentare il modello a oggetti di analisi (*Analysis Model*). Anche per questo modello viene presentata una tecnica utile per la sua realizzazione, una serie di consigli, nonché i vari tranelli in cui si può correre il rischio di imbattersi. Nei progetti reali non è sempre possibile realizzare il modello di analisi, per via dei soliti problemi (mancanza di tempo, personale, ecc.), pertanto si fornisce una serie di consigli su come ridurre il rischio generato dalla mancata realizzazione, su quando sia possibile trascurarlo, su quando invece è opportuno realizzarlo, ecc. Prima di proseguire nella presentazione del modello successivo, viene presentato un formalismo di importanza storica per il processo di realizzazione dei vari modelli a oggetti che tuttora conserva delle aree di applicazione, ossia il metodo delle *CRC Cards*.

L'ultimo modello a oggetti presentato, probabilmente il più noto, è quello di disegno (*Design Object Model*). Nell'ambito di ogni iterazione, dovrebbero esistere almeno due versioni di modello a oggetti di disegno: quello di specifica e quello di implementazione. Il primo dovrebbe essere realizzato prima dell'inizio della codifica, mentre il secondo è il risultato dell'evoluzione che il disegno subisce durante l'implementazione (variazioni in corso d'opera). Come al solito si riportano immancabili esempi, consigli su come realizzarlo ecc. Il capitolo si conclude cercando di rispondere alla fatidica domanda: "Quando un modello a oggetti di disegno si può considerare ben costruito?".

Capitolo 9 – Diagrammi di interazione

Le parti che costituiscono un qualsivoglia sistema non sono assemblate per persistere semplicemente in una posizione stabile, bensì interagiscono tra loro scambiandosi messaggi al fine di raggiungere uno scopo ben preciso. Pertanto la descrizione di ogni comportamento necessita di essere modellata sia attraverso una prospettiva che mostri la struttura statica degli elementi cooperanti, sia mediante un'altra che illustri il relativo comportamento dinamico. Obiettivo del presente capitolo — e di quello successivo — è illustrare i formalismi forniti dallo UML per modellare comportamenti dinamici.

In particolare nel capitolo nono l'attenzione è focalizzata sui diagrammi di interazione (*interaction diagram*). Con questo nome si indicano i diagrammi di sequenza (*sequence diagram*) e quelli di collaborazione (*collaboration diagram*). Quantunque permettano di mostrare lo stesso contenuto informativo, si diversificano per l'aspetto dell'interazione cui è attribuita maggiore importanza: i diagrammi di sequenza enfatizzano lo scambio dei messaggi nel tempo, mentre i diagrammi di collaborazione attribuiscono maggiore importanza all'organizzazione degli "oggetti" coinvolti nell'interazione modellata.

L'utilizzo di questi diagrammi comprende quasi tutte le fasi dei processi di sviluppo del software: si va dall'analisi dei requisiti in cui possono essere utilizzati per illustrare gli scenari dei casi d'uso (soprattutto i diagrammi di sequenza), alla fase di analisi e disegno in cui mostrano le dinamiche interne del sistema.

Si tratta di diagrammi ampiamente utilizzati nei capitoli precedenti e che, nel capitolo nono, trovano un'illustrazione completa. Di questi due diagrammi è presentato in modo approfondito il formalismo, consigli su come e quando utilizzarli, ecc. Il tutto sempre coadiuvato da un consistente insieme di esempi nonché dalle sezioni dedicate ai consigli relativi allo stile.

Capitolo 10 – Le attività di stato

In questo capitolo viene conclusa l'illustrazione dei diagrammi dello UML destinati a modellare comportamenti dinamici. In particolare sono illustrati i diagrammi degli stati (*state chart diagram*) e quelli di attività (*activity diagram*). Anche in questo caso si tratta di diagrammi ampiamente utilizzati nel corso dei capitoli precedenti che però nel capitolo decimo sono affrontati in maniera organica.

I diagrammi degli stati descrivono il ciclo di vita di automi a stati finiti. Più precisamente, possibili sequenze di stati e azioni attraverso le quali istanze di opportuni elementi possono procedere durante il relativo ciclo di vita. Tale evoluzione è l'effetto delle reazioni innescate in tali istanze al verificarsi di opportuni eventi discreti. I diagrammi di attività (discendenti dei celebri diagrammi di flusso, *flowchart diagram*) sono una variante di quelli degli stati, ove gli stati rappresentano l'esecuzione di azioni o sottoattività e le transizioni sono innescate dal completamento di tali azioni o sottoattività.

Questi diagrammi possono prevedere un considerevole utilizzo nelle fasi di analisi dei requisiti, che, tipicamente, si affievolisce nelle restanti fasi del processo di sviluppo del

software. I diagrammi delle attività trovano grande applicazione nella fase di analisi dei requisiti dove sono impiegati per mostrare gli scenari dei casi d'uso. Nelle restanti fasi tendono a essere utilizzati qualora vi sia la necessità di mostrare comportamenti concorrenti e la precisa allocazione delle responsabilità. Sempre i diagrammi delle attività recitano un ruolo di primissima importanza ogniqualvolta sia necessario illustrare graficamente processi e sottoprocessi concorrenti.

I diagrammi degli stati sono utilizzati indistintamente in tutte le fasi del processo di sviluppo del software qualora vi sia la necessità di illustrare il ciclo di vita di "oggetti" (anche composti) che possono evolvere attraverso un insieme finito di stati.

Di questi due diagrammi è illustrato approfonditamente il formalismo, consigli su come e quando utilizzarli, ecc. Il tutto sempre coadiuvato da un consistente insieme di esempi nonché dalle sezioni dedicate ai consigli relativi allo stile grafico.

Capitolo 11 – Anche gli aspetti "fisici" sono importanti

Il libro si conclude con la presentazione dei diagrammi forniti dallo UML atti a modellare aspetti fisici del sistema: i diagrammi di implementazione (*implementation diagram*). Con questo termine ci si riferisce ai diagrammi dei componenti (*component diagram*) e a quelli di dispiegamento (*deployment diagram*).

I diagrammi dei componenti mostrano appunto la struttura dei componenti di cui è composto il sistema, inclusi gli elementi che li specificano (classificatori) e i manufatti che li implementano. I diagrammi di dispiegamento mostrano la struttura dei nodi che costituiscono l'architettura fisica del sistema e nei quali vengono installati i componenti. Questi diagrammi si prestano anche a utilizzi più estesi. Nella modellazione dell'area business, per esempio, i componenti mostrano procedure di business e manufatti, mentre i nodi rappresentano organizzazioni e risorse del business. Sebbene costituiscano il modello fisico, la loro produzione inizia già dalle primissime fasi dell'analisi dei requisiti, poiché considerazioni di carattere architetturale forniscono importantissimi feedback allo studio di fattibilità dei requisiti.

Di questi diagrammi, in particolare di quello dei componenti, sono mostrate importanti lacune (quasi completamente riassorbite dalla versione 2 di UML), è illustrato ampiamente il formalismo, sono forniti consigli su come e quando utilizzarli, un consistente insieme di esempi, vengono proposte indicazioni relative allo stile grafico, ecc.

Il capitolo si conclude con la presentazione di un processo dimostratosi particolarmente valido nella progettazione di sistemi basati sui componenti.

Appendice A – UML e i linguaggi di programmazione non OO

Sebbene UML sia un linguaggio di modellazione ideato per la progettazione di sistemi OO e component-based, si presta a essere utilizzato per la produzione di sistemi che prevedano linguaggi di programmazione non basati su tali paradigmi. Ciò è vero utiliz-

zando opportune cautele. Nell'Appendice A sono illustrati utili criteri, idee e problematiche relative all'utilizzo dello UML con linguaggi di programmazione non OO.

Appendice B – UML e la modellazione di basi di dati non OO

Obiettivo dell'appendice B è illustrare una sorta di profilo utilizzabile per modellare basi di dati fondate su paradigmi non OO. In particolare l'attenzione è focalizzata sulle basi di dati relazionali, sebbene i concetti esposti si prestino a essere estesi anche a basi di dati fondate su diversi paradigmi.

Appendice C – Il profilo EJB

L'appendice C è dedicata ad una breve illustrazione di un profilo (non standard) utilizzabile per la modellazione di sistemi component-based fondati sull'architettura *Sun EJB*. Brevemente, un profilo è una particolare estensione dello UML, o meglio una collezione di estensioni predefinite, le quali nascono dall'esigenza di standardizzare l'utilizzo dello UML per domini, scopi o tecnologie ampiamente diffuse.

Appendice D – Glossario

Un utile glossario di riferimento costituisce l'Appendice D.

Appendice E – Bibliografia ragionata

Un elenco di libri, riviste e siti web di riferimento.

Ringraziamenti

Come ogni libro che si rispetti anche qui è presente l'immancabile e doverosa sezione dedicata ai ringraziamenti.

In primo luogo la personale riconoscenza dell'autore va al fantastico team che lo ha assistito e coadiuvato durante tutto il ciclo di sviluppo del libro. In particolare si ringraziano gli amici Roberto Virgili, (`www.RobertoVirgili.com`, team leader architetto, nonché sviluppatore di sistemi distribuiti, vero e proprio formatore di menti che, tra l'altro, ha sostenuto per primo l'idea del libro: i lettori sanno quindi a chi rivolgersi per eventuali ritorsioni), e Antonio Rotondi (`Antonio.Rotondi@Artech.freeserve.co.uk`, team leader, architetto di sistemi distribuiti e infrastrutture di sicurezza che negli ultimi anni ha lavorato intensamente allo sviluppo di sistemi per la gestione di smart card). Si tratta di due tecnici informatici di profonda levatura ed esperienza, veri e propri profeti dell'arte di realizzare sistemi informatici che funzionano.

Ringraziamenti aggiuntivi spettano a Gianluca Pignalberi, G.Pignalberi@tiscali.it, libero professionista, esperto di metodi di Intelligenza Artificiale applicati all'elaborazione delle immagini e di Linguistica Computazionale) sempre prodigo di consigli e preziosi suggerimenti, specie relativi alla sfera delle strutture dati e dei linguaggi formali.

Un particolare ringraziamento va poi a Francesco Saliola che ha curato la redazione e l'impaginazione del libro.

Ulteriori ringraziamenti spettano a Giovanni Puliti (Java enthusiast, esperto di Java e tecnologie annesse, scrittore, articolista e direttore della rivista web www.MokaByte.it) e a Claudio Bergamini (amministratore di profonda conoscenza tecnica e rara lungimiranza, www.imolinfo.it) che hanno creduto nel libro e si sono adoperati proficuamente affinché il progetto potesse vedere la luce di Internet.

In questo gruppo di ringraziamenti va sicuramente annoverata Hops Libri che ha dimostrato, al solito, notevole coraggio decidendo di pubblicare un corposo volume che, probabilmente contro corrente, affronta tematiche non sempre popolari come la progettazione formale di sistemi OO (o component-based che dir si voglia).

Altri ringraziamenti doverosi spettano ai familiari dell'autore e a tutti i **veri** amici, presenti nei tanti momenti di bisogno.

La profonda riconoscenza e il ringraziamento dell'autore spettano poi a tutti coloro che accoglieranno l'invito di leggere il libro e a fornire le proprie riflessioni, al fine di arricchire il presente lavoro rendendolo sempre più vicino alle tipiche esigenze dei progetti reali.

Dulcis in fundo — forse — un ringraziamento "particolare", molto particolare, va ai sedicenti tecnici, i vari "gatto" e "volpe" onnipresenti in tutte le organizzazioni. A tutti coloro che, chiusi nella loro arroganza, si sentono perpetuamente più furbi, "patrigni" dispensatori di consigli sempre troppo gratuiti. Agli adepti delle ditte a "conduzione familiare", che obbligano giovani professionisti a combattere contro muri di gomma, a lavorare sempre di più e a dare costantemente il meglio di sé fino alla nausea per dimostrare che la qualità non alberga esclusivamente nei libri di teoria… Ammesso che queste persone siano in grado di capirla!

Scuse

Le personali scuse dell'autore vanno ai teorizzatori dell'informatica, agli autori dei vari testi sacri, e in primo luogo ai Tres Amigos. Il loro contributo alla semplificazione del processo di sviluppo dei sistemi informatici è così evidente da non richiedere assolutamente commenti. Si tratta di vere oasi di formazione intellettuale nel deserto degli "Azzeccagarbugli", di fari che illuminano la rotta di tutti coloro per i quali l'informatica è una scienza… Ciò nonostante si è ritenuto opportuno non prendersi troppo sul serio, permettendosi ironie anche nei confronti dei mostri sacri — non solo nel senso traslato del

termine — dell'informatica… D'altronde i cimiteri sono gremiti di lapidi che declamano: "Qui giace una persona insostituibile".

Le scuse personali dell'autore sono rivolte ai lettori più attenti, se, talune volte, l'impeto di voler illustrare concetti complessi nel modo più semplice possibile possa aver travolto la trattazione erudita e portato a "frasi fuorvianti". Ciò nonostante si è deciso di perseverare diabolicamente in tale direzione confidenti che "quando il saggio indica la luna solo lo stolto fissa il dito".

Le scuse finali sono relative alla voluminosità del libro… attualizzando una frase che Voltaire usava scrivere nelle lettere alla sua amica: "… vi porgo le mie scuse per la voluminosità della lettera, ma proprio non ho avuto tempo per scriverne una concisa".

Breve biografia dell'autore

Luca Vetti Tagliati è nato a Roma il 03/12/1972 (ottima annata per il vino). Ha da sempre studiato IT: quando all'asilo bucava i fogli di carta con la matita, manifestava i primi sintomi da programmatore di schede perforate.

Ha iniziato a lavorare professionalmente nel 1991 occupandosi di sistemi firmware ed assembler per microprocessori della famiglia iAPXx286.

Nel 1994/95 ha intrapreso il lungo cammino nel mondo Object Oriented grazie al meraviglioso linguaggio C++. Nel 1996 si è trovato catapultato nel mondo Java.

Negli ultimi anni ha applicato la progettazione/programmazione Component Based in settori che variano dal mondo delle *smart cards* (collaborazione con la Datacard, `www.datacard.com`) a quello del trading bancario (`www.hsbc.com`).

Ha lavorato nella sede londinese della banca HSBC con funzioni di chief designer, team leader e membro del gruppo PEG (Processing Engineering Group) istituito per stabilire un processo di sviluppo standard da applicarsi per lo sviluppo di sistemi software. Il PEG si è avvalso della collaborazione di personaggi del calibro di John Daniels [BIB15] e Frank Armour [BIB14].

Attualmente lavora come consulente per importanti gruppi finanziari ubicati nella City di Londra

Convenzioni grafiche

Il carattere senza grazie, accoppiato a un'opportuna icona, è utilizzato per evidenziare parti di testo meritevoli di particolare attenzione oppure dedicati esclusivamente a un pubblico esperto.

L'icona della lampadina, congiuntamente al carattere senza grazie viene utilizzata per mettere in luce un determinato concetto chiave, un consiglio pratico o un altro argomento che meriti particolare attenzione.

L'icona del pericolo generico viene utilizzata per evidenziare sezioni, parti, note o paragrafi aventi una difficoltà intrinseca elevata e dedicati a un pubblico esperto. Si tratta comunque di paragrafi completamente opzionali la cui lettura e comprensione non sono assolutamente indispensabili per l'apprendimento di quanto riportato nei paragrafi e capitoli successivi. Ad ogni modo, non scarseggiano mai ampie descrizioni testuali atte a rendere accessibili a tutti anche le trattazioni più complesse.

L'icona degli ingranaggi è stata utilizzata per evidenziare sezioni, parti note o paragrafi dedicati ai processi di sviluppo del software. Chi fosse allettato unicamente dal linguaggio dello UML, e quindi è interessato solo fino a un certo punto ai processi di sviluppo, può tranquillamente disinteressarsi di questi paragrafi.

L'icona della tavolozza dei colori evidenzia paragrafi contenenti linee guida per il miglioramento dell'eleganza e della comprensibilità dei diagrammi. L'obiettivo è pertanto migliorare lo stile senza entrare nel merito della semantica — in altre parole si focalizza l'attenzione sullo strato di presentazione — sebbene poi le due caratteristiche non possano essere trattate in maniera completamente disgiunta.