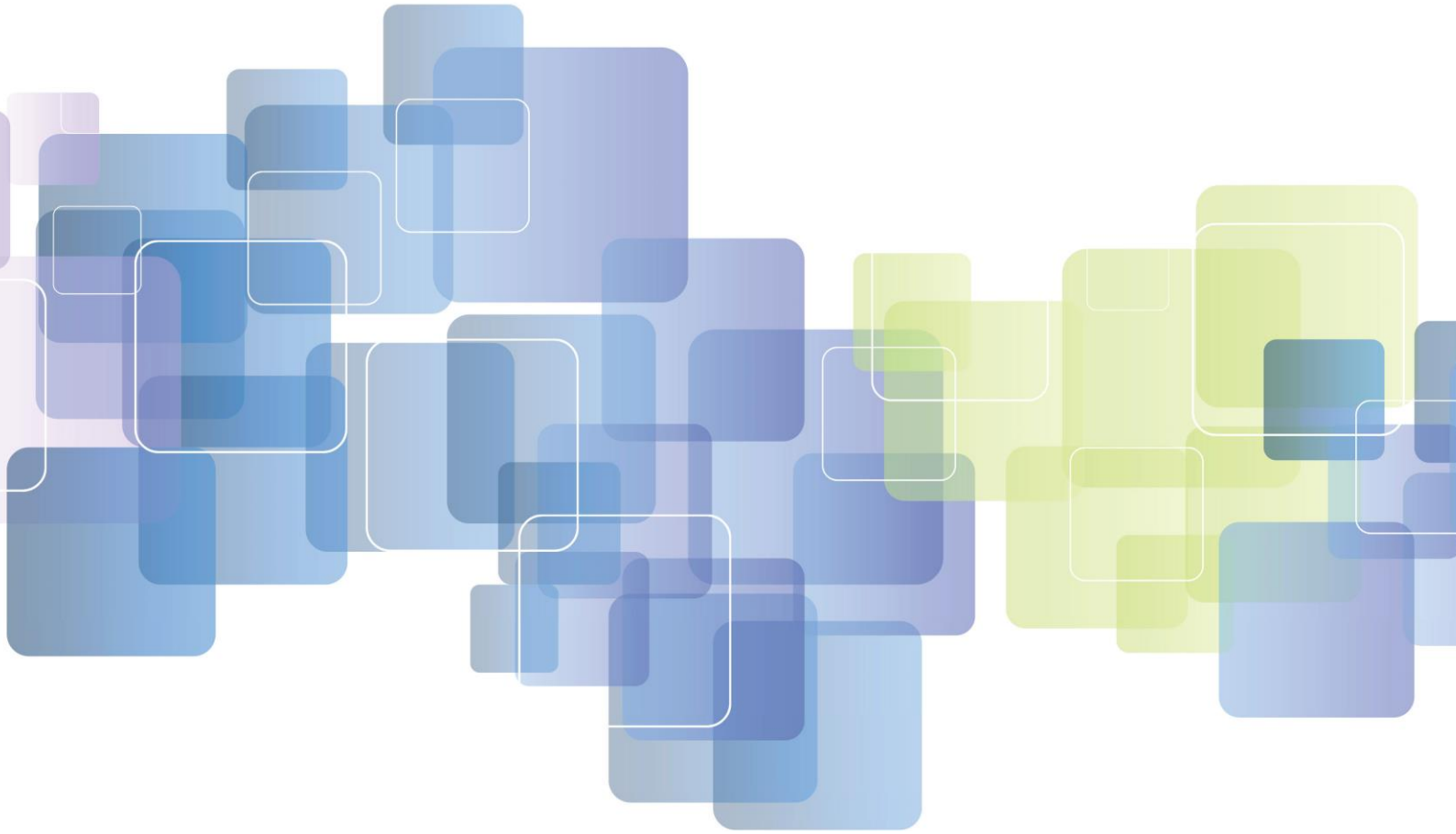
TecnetDati

MDA (Model-Driven Architecture) e UML:

costruire sw partendo dai modelli

Iolanda Salinari e Sandro Bellu



Si tratta di costruire le applicazioni portandosi ad un livello di astrazione più alto. Tuttavia, ...

Durante l'ultimo OMG Information Day, partendo dalle problematiche d'integrazione dei sistemi aziendali, è stato presentato l'ormai noto approccio allo sviluppo del software chiamato MDA (Model-Driven Architecture) in relazione all'evoluzione dei linguaggi di modellazione del software (UML) e dei tools che permettono di progettare e realizzare applicazioni basate su questi standard.

Molte aziende hanno investito notevoli risorse in sistemi di vario genere: CRM, Web, ERP ed altri; spesso però non ci si è soffermati su come questi sistemi potessero lavorare sinergicamente fra loro su informazioni condivise ed integrate. In un mondo perfetto le informazioni cruciali per l'azienda dovrebbero essere univoche e disponibili in modo veloce, migliorando l'efficienza dell'azienda e di conseguenza la soddisfazione del cliente, ma l'integrazione è un compito tutt'altro che semplice: lo scenario è in continua evoluzione sia in termini di esigenze di business da un lato che di tecnologie a disposizione dall'altro.
I sistemi informativi aziendali devono essere messi in grado di far fronte in maniera efficace ed efficiente ai cambiamenti dell'azienda e del mercato e di sfruttare appieno le potenzialità dell'innovazione tecnologica.

L'OMG (Object Management Group [http://www.omg.org/]), in collaborazione con varie entità del panorama ICT mondiale (IBM, Computer Associates, Borland, HP, META Group, ...), è diventato ormai da tempo un punto di riferimento determinante per lo sviluppo degli standard atti a rendere la progettazione / realizzazione del software un fenomeno controllabile, meno artigianale e pronto a recepire le evoluzioni tecnologiche e di business.

Per vedere di cosa si tratta, affronteremo nei prossimi paragrafi, tre argomenti strettamente correlati:

MDA, ovvero "devi costruire applicazioni o scrivere linee di codice?"

L'idea di MDA nasce con l'evoluzione dei linguaggi di modellazione visuale del software, principalmente con UML.
La Model-Driven Architecture intende appunto fornire un quadro concettuale, il più possibile esauriente, per un approccio allo sviluppo basato sui modelli.
Ma qual è l'attuale stato dell'arte di tale approccio?
Mentre è ormai una prassi comune utilizzare modelli quali l'Entity/Relationship per la progettazione di basi di dati relazionali, non è ancora così diffusa la modellazione del

comportamento del sistema, delle relazioni funzionali fra le diverse componenti software e di altri aspetti.

Spesso lo scopo della modellazione è inteso, in modo riduttivo, a fornire semplicemente una documentazione formale dell'applicazione; inoltre tutto ciò che "non è programmazione" è visto ancora con sospetto, e non solo dagli sviluppatori. In realtà, lo sviluppo per modelli dovrebbe consentire di concentrarsi maggiormente nel fornire soluzioni alle problematiche più impegnative dello sviluppo piuttosto che su problemi di più basso livello, quali ad esempio errori di sintassi su un determinato linguaggio. In questo senso, il "nuovo" approccio dovrebbe risultare più stimolante anche per lo stesso sviluppatore, ponendolo maggiormente in un ruolo di risolutore di problemi anziché relegarlo in quello di esperto di un certo linguaggio di programmazione.

Alla base dello sviluppo per modelli è l'Unified Modeling Language, un linguaggio formale capace di descrivere "**cosa**" fa l'applicazione, prima ancora di capire "**come**" lo fa.

I modelli previsti da UML forniscono una sorta di "template" che ci guida nella costruzione del sistema, assolvendo più compiti durante le varie fasi dello sviluppo:

- consentono di verificare che un sistema soddisfi i requisiti di business
- rendono il sistema comprensibile (se si conosce il linguaggio di modellazione)
- permettono di far evolvere il sistema in modo controllato, seguendo le necessità del business; agevolano quindi la manutenzione
- rappresentano il dominio dell'applicazione, per quanto complessa, multipiattaforma, ecc...
- se indipendenti dalla piattaforma tecnologica, salvaguardano gli investimenti nel tempo, mentre lo scenario tecnologico cambia.

Una delle finalità più importanti di un modello è quella di facilitare la comunicazione tra le diverse parti coinvolte nel ciclo di vita del software (chi indaga i requisiti, chi esegue l'analisi del sistema, chi sviluppa, chi effettua le attività di testing). Ogni parte lavora su una vista diversa dello stesso sistema, ma le differenti viste devono essere coerenti e condividere una base comune di costrutti e di conoscenze. Un linguaggio di modellazione grafica come l'UML agevola la possibilità di una visione comune del sistema da parte di tutti gli addetti ai lavori, non da ultimo da parte dei clienti e del management aziendale.

Un altro fattore da considerare è che i linguaggi di modellazione sono diventati sufficientemente potenti da consentire di specificare il sistema ad un livello di dettaglio tale da rendere possibile, con il supporto degli strumenti idonei, la generazione di gran parte del codice richiesto da un sistema.
E' possibile quindi simulare il funzionamento del sistema "eseguendo" i modelli, controllare la correttezza del sistema anticipando le attività di verifica e validazione a livello di modello, con evidenti vantaggi.

Il software nasce da un'esigenza che "magicamente" (a volte sembra proprio così!) viene tradotta in linee di codice che la soddisfano, ma come gestire in modo coerente e tracciabile la trasformazione che in qualche modo genera software a partire da un'idea, spesso in fase embrionale ed incompleta? Se fra i requisiti dell'applicazione e le linee di codice non esiste un "ponte" capace di assicurare la coerenza di questo passaggio,

diventa molto difficoltoso garantire un'evoluzione controllata del sistema a fronte di nuovi requisiti.
Molto spesso le cose si complicano ulteriormente, perché è necessario integrare più sistemi e fare in modo che possano interoperare in modo efficace. Per garantire l'integrazione e l'interoperabilità è indispensabile conoscere in modo chiaro le funzioni logiche delle applicazioni; se non esiste un modello che renda comprensibile il comportamento dell'applicazione e le scelte che ne hanno determinato una specifica implementazione oppure il software non è implementato in modo coerente rispetto al modello, probabilmente si renderanno necessari interventi il re-engineering del codice ed il supporto di chi ha scritto o conosce il software esistente. In ogni caso si tratta di interventi il cui costo cresce esponenzialmente con le dimensioni e la complessità dell'applicazione.

MDA si propone come la soluzione per creare questo "magico ponte" tra requisiti e software, soprattutto grazie al linguaggio di modellazione UML, agli strumenti che lo supportano e alle best practices rappresentate dai patterns.
Si tratta di descrivere interamente un'applicazione attraverso modelli, che supportino ogni fase del ciclo di vita del software; tali modelli devono essere:

- espressi in modo formale
- collegati l'uno con l'altro in modo non equivoco
- sufficientemente flessibili.

Ma l'obiettivo di MDA è ancora più ambizioso: fornire uno standard completo per la creazione di modelli indipendenti dall'implementazione, in modo che possano essere "mappati" su qualsiasi piattaforma, presente o futura.
Il cuore di MDA è un insieme di standard (MOF, UML, CWM e XMI) e di tecnologie (CORBA, .NET, Java e così via); a partire da tali standard e servizi, è possibile costruire profili specifici di dominio per la finanza, l'e-commerce, ecc... e realizzare, per ognuno di questi domini, specifiche applicazioni conformi agli standard di supporto.

MDA separa i due elementi fondamentali di un'applicazione in due distinti modelli:

- PIM, il modello che definisce che definisce le funzionalità di business ed il comportamento, ovvero l'essenza del sistema, indipendentemente dalla sua implementazione tecnologica
- PSM, il modello specifico di piattaforma, che consente di "mappare" il PIM su una specifica tecnologia (Java, CORBA, ...), senza alterare il PIM stesso.

La divisione dei due modelli supporta l'interoperabilità; una funzione di business non ha necessità di conoscere l'implementazione di un'altra funzione di business per potere accedere ai suoi servizi: l'interfaccia è definita secondo le modalità del PIM, indipendentemente dalla tecnologia.
Di conseguenza nonostante le continue modifiche all'ambito tecnologico, le funzionalità di business possono rimanere relativamente stabili e non subire particolari impatti da tali cambiamenti.

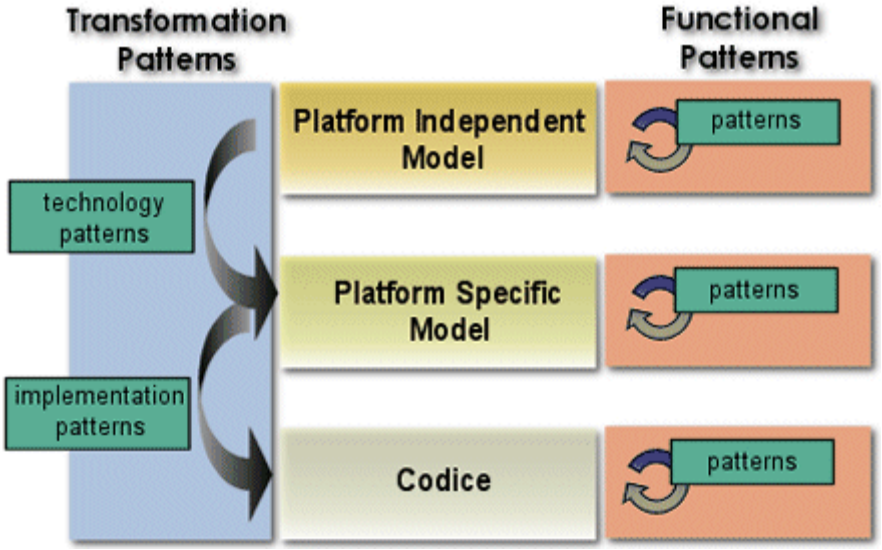
MDA suddivide lo sviluppo del software in 3 passi principali:

1. creazione del Platform Independent Model (PIM)

2. trasformazione di PIM in Platform Specific Model (PSM)
3. generazione del Software

La definizione di PIM e PSM si avvale di Patterns predefiniti, personalizzabili e riusabili, così come la trasformazione PIM à PSM (patterns tecnologici) e la generazione del software (patterns di implementazione) a partire dal PSM.

In sostanza, tutto quello che può essere descritto in PIM, può essere trasformato in PSM; tutto quello che è stato descritto con PSM può generare codice.



Una volta inserite tutte le informazioni necessarie nel modello, gli strumenti possono generare gran parte del codice necessario ad un sistema, grazie alla potenza del linguaggio di modellazione, che permette di descrivere il comportamento del sistema ad un sufficiente livello di granularità.
Ma MDA non è solo generazione automatica del codice, è soprattutto un approccio allo sviluppo del software meno artigianale e più formale, che consente una visione comune del sistema ad analisti, architetti, programmatori, addetti al testing, ecc...
Questo tipo di approccio varia notevolmente la prospettiva con la quale ci si pone di fronte allo sviluppo di applicazioni: permette di focalizzare l'attenzione degli sviluppatori su "cosa" l'applicazione deve fare, piuttosto che sui dettagli di implementazione; consente inoltre una tracciabilità più rigorosa tra requisiti e prodotti dello sviluppo.

Generare il software a partire dai modelli non è ovviamente gratuito, richiede un impegno e un rigore notevoli nelle fasi di analisi e disegno, la padronanza dei concetti e delle tecniche che UML sottintende e la conoscenza delle potenzialità che tale linguaggio di modellazione può offrire.

Il nuovo UML: Unified Modeling Language 2.0

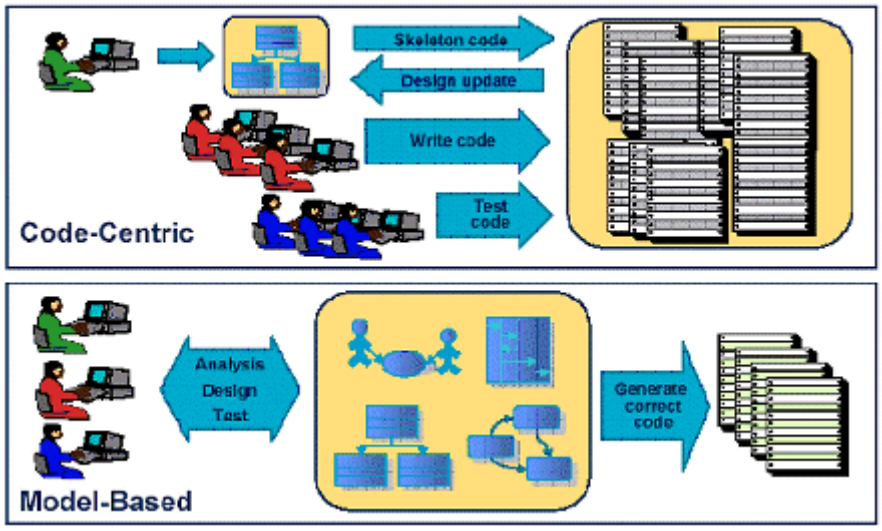
UML è il linguaggio di modellazione per specificare, costruire, visualizzare e documentare gli artefatti di un sistema.
Nasce tra il '94 e il '95 ad opera di tre nomi illustri nel campo dello sviluppo OO: Grady Booch, James Rumbaugh e Ivar Jaobson; raccoglie quindi le caratteristiche principali delle metodologie dei rispettivi padri oltre a numerosi contributi di altri metodologi. Grazie alla sponsorizzazione dei principali vendors ICT (IBM, HP, Microsoft, Oracle, ...), nel novembre del '97, UML è stato adottato come linguaggio standard dell'OMG. UML è un linguaggio in continua evoluzione per rispondere alle varie esigenze che emergono nel settore ICT (sia da parte delle aziende che sviluppano software che dai vendors di tools di supporto). Esistono a questo scopo procedure controllate e gruppi di lavoro che hanno il compito di valutare le esigenze e di tradurle in nuovi standard, cercando ovviamente di non stravolgere le precedenti versioni.

L'ultima versione di UML (2.0) ha essenzialmente lo scopo di colmare alcune lacune delle versioni precedenti e di utilizzare pienamente l'approccio MDA.

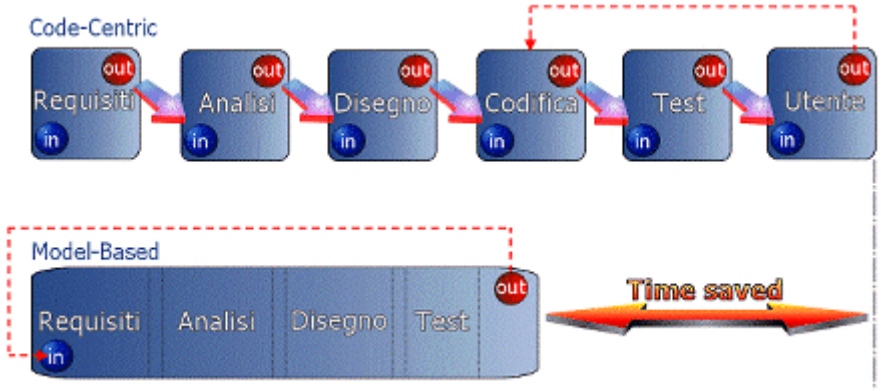
Alcuni tra i principali obiettivi di UML 2.0 sono stati:

- migliorare i meccanismi di estensione
- arricchire le specifiche architetturali
- potenziare il supporto per lo sviluppo basato sui componenti
- migliorare la gestione degli eventi e dei flussi dati nei diagrammi di attività
- fornire una notazione per i patterns.

L'immagine sottostante mostra come l'approccio basato sui modelli (Model-Driven o Model-Based) si differenzia dall'approccio basato sulla scrittura del codice (Code-Centric):

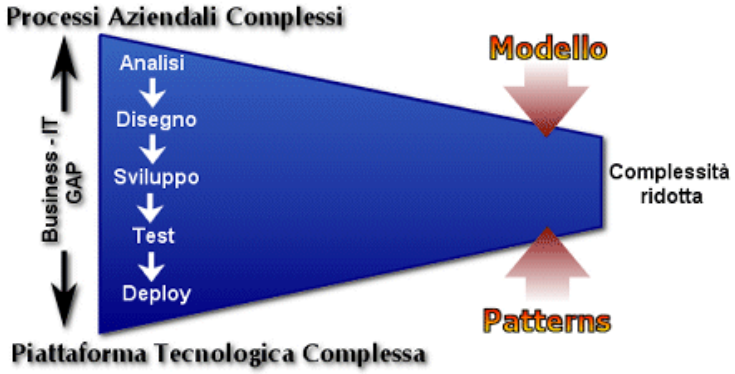


Il ciclo di vita del software con UML e MDA, diventa così:



Tools: gli strumenti al servizio dell'approccio per modelli

L'adozione di MDA e UML richiede l'uso di strumenti di supporto. Gli strumenti devono essere ovviamente adatti a svolgere bene il loro compito. Spesso, purtroppo, i vendors promettono molto più di quello che lo strumento può fare e i clienti, da parte loro, si aspettano troppo dallo strumento, sperando che sia la soluzione ad ogni problema.
Dev'essere chiaro che gli strumenti possono semplicemente agevolare il lavoro, renderlo più efficiente, ma occorre investire, prima ancora che sugli strumenti, sulla formazione, relativamente ai concetti, alle tecniche di analisi / disegno OO e alle best practices (patterns), e sull'applicazione di tali conoscenze al lavoro quotidiano (inizialmente anche senza il supporto di tools adeguati) per acquisire via via un'esperienza reale.



In assenza delle conoscenze opportune, anche con la disponibilità dei linguaggi più evoluti e degli strumenti più sofisticati, è possibile produrre software di scarsa qualità o comunque non sfruttare appieno i benefici dell'approccio Object Oriented.

Considerazioni

Quando dai linguaggi macchina si è passati ai linguaggi compilati, come il Cobol o il C, qualcuno sospirando diceva: "io non so come viene compilato questo programma, come questa istruzione C si trasforma in istruzioni eseguibili dalla macchina. Funzionerà?". Lo stesso accadeva con i primi strumenti capaci di tradurre diagrammi Entity/Relationship in DDL. Oggi credo che nessuno scriva più DDL completamente "a mano" ed i linguaggi macchina sono utilizzati solo per applicazioni di nicchia altamente specializzate.

Si tratta "semplicemente" di costruire applicazioni, portandosi ad un livello di astrazione più alto. E' il concetto sul quale, ad esempio, si basano tutti i linguaggi compilati, la modellazione dei dati per la generazione degli schemi di database, sul quale si basano anche UML ed MDA, dove l'implementazione avviene a partire dal linguaggio di modellazione.

Qualcuno profetizza che il divario tra utente e programmatore sta via via diminuendo e che, con tutta probabilità, un giorno scomparirà ... staremo a vedere. Di sicuro oggi esistono piattaforme (Java, .NET, ...) che rispondono ai requisiti dell'approccio object-oriented come mai prima d'ora, esistono linguaggi di modellazione "potenti" (UML), strumenti efficaci che sono in grado di mettere in pratica le promesse dell'OO.

L'uso corretto di MDA/UML facilita senz'altro la produzione di software di qualità; ma occorre valutarne l'applicabilità situazione per situazione (non tutte le applicazioni hanno lo stesso grado di complessità o presentano problematiche di integrazione/interoperabilità; in alcune realtà aziendali è già consolidato l'uso di particolari tecniche e processi di sviluppo).

MDA/UML non è la panacea che risolve tutti i mali. Non è un bottone magico che dal nulla crea immediatamente l'applicazione del secolo. E' qualcosa capace di portare ad un livello di astrazione più alto lo sviluppo del software: questo, come in passato nei casi citati ed in altri, permette di dominare la complessità, di avere tempi di risposta più rapidi ed una più chiara comunicazione fra le persone. Consente ai sistemi di evolvere di pari passo con le tecnologie e con l'azienda; salvaguarda gli investimenti in quanto non li lega ad una specifica piattaforma software, ma descrive i sistemi in modo indipendente dal contesto tecnologico.

Per chi ha vissuto l'era dei sistemi CASE della prima (e seconda) generazione tutto ciò non rappresenta una vera novità. Dopo l'insuccesso di progetti come AD-Cycle molte aziende avevano rivisto le loro aspettative nei confronti di un approccio integrato allo sviluppo e alla manutenzione del software. Il successivo avvento di Internet ha evidenziato nuove priorità e nuovi modelli (per esempio FAD) che riportavano in primo piano la tempestività di risposta, anche a scapito della pulizia teorica delle soluzioni adottate. Oggi i tempi sembrano essere maturi per ripensare il processo di sviluppo, anche alla luce di nuove tecniche e nuovi strumenti.
Tuttavia ... mi viene in mente il titolo di un vecchio film di Woody Allen: "Provaci ancora Sam".

Glossario

MOF
Meta-Object Facility, è un *metamodello*, che definisce i concetti base per costruire un modello e memorizzarlo in un repository.
I due linguaggi di modellazione UML e CWM sono costruiti a partire dai costrutti basilari di MOF. MOF, fornendo un metamodello comune, supporta la portabilità e lo scambio di una serie di metamodelli tra diversi repository (ad es. è possibile scambiare modelli UML tra tools di differenti vendors, oppure portare un modello a oggetti UML su un tool di data-modeling e viceversa).

CWM
Il Common Warehouse Model definisce un metalinguaggio, derivato da MOF, per descrivere i data warehouse e i sistemi correlati.

XMI
XML Metadata Interchange definisce un mapping da UML a XML, rendendo possibile convertire un modello UML in formato XML, distribuirlo e riconvertirlo in formato UML. In questo modo è possibile scambiare modelli UML tra tools e piattaforme diverse. Il mapping di XMI utilizza i metadati di MOF.

Meccanismi di estensione
I meccanismi di estensione di UML (stereotipi, nuove proprietà, vincoli) permettono al modellista di modificare la semantica degli elementi del metamodello. Consentono di creare di nuovi blocchi costruttivi, nuove relazioni, o di modificare le caratteristiche di quelli esistenti, per far fronte ad esigenze specifiche di modellazione.

Profili UML
Si tratta di collezioni di stereotipi, proprietà e regole che specializzano lo standard UML, per utilizzarlo in modo più efficace nella costruzione di modelli di sistemi ed entità in domini specifici (es. Profilo per il Business Modeling, Profilo per il Web Modeling)
I nuovi profili possono essere creati grazie ai meccanismi di estensione di UML.

Patterns
I Patterns sono soluzioni standardizzate (e sperimentate) di situazioni ricorrenti dalle caratteristiche precise. Rappresentano quindi le best practices per risolvere problemi ricorrenti. Sono applicati soprattutto nel campo del disegno del software, ma esistono anche patterns di analisi, patterns architetturali, ecc...

DDL
Data Definition Language, linguaggio di definizione dei dati, messo a disposizione dai DBMS.

FAD
Frantic Application Development, sviluppo di applicazioni *frenetico*.
Frantic dà l'idea di qualcosa di frenetico, concitato, convulso, quasi delirante.
(Vedi articolo di Ed Yourdon) .

Tecnet Dati s.r.l.
C.so Svizzera 185 –
10149 - Torino (TO), Italia
Tel.: +39 011 7718090 Fax.: +39 011 7718092
P.I. 05793500017 C.F. 09205650154
www.tecnetdati.com