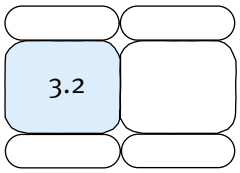
PROGRAMMAZIONE ORIENTATA AGLI ASPETTI: SCENARI DI ADOZIONE INDUSTRIALE

Filippo Diotalevi

Il grande successo della programmazione orientata agli oggetti non ha limitato la ricerca di nuovi paradigmi e tecnologie che possano incrementare la produttività degli sviluppatori e la qualità del software. In questi ultimi anni, la programmazione orientata agli aspetti sta conoscendo una crescente diffusione. Questo articolo è sia un'introduzione a questo nuovo paradigma di programmazione che una guida alla sua graduale adozione nello sviluppo di applicazioni aziendali.



1. INTRODUZIONE

L'evoluzione dell'informatica ha portato all'affermarsi della programmazione orientata agli oggetti [1] come principale strumento per lo sviluppo di applicazioni aziendali. Complementariamente a questa affermazione si è assistito al nascere e allo svilupparsi di metodologie, linguaggi di analisi e design e di strumenti orientati agli oggetti che hanno svolto un ruolo fondamentale nel permettere di costruire sistemi informativi sempre di maggior complessità, fino ad arrivare ai moderni software distribuiti, multipiattaforma e caratterizzati dai più alti standard di scalabilità e sicurezza.
La realizzazione di applicazioni che risolvono tali problematiche ha però comportato un aumento di pari portata della complessità del software; infatti, le tematiche e le tecnologie che i professionisti del settore devono padroneggiare per poter rispondere adeguatamente alle richieste del mercato si sono moltiplicate, rendendo di conseguenza il processo di produzione del software più critico e complicato.
Se da un lato questa crescente complessità è implicita nella richiesta di funzionalità evolute ed alti livelli di servizio, è stato osservato più volte [2] che essa è anche conseguenza del fatto che il paradigma di programmazione orientato agli oggetti non si è dimostrato capace negli anni di modellare in modo adeguato tutte le problematiche emerse. La scoperta di queste limitazioni ed inefficienze ha stimolato di conseguenza la nascita di nuove teorie e proposte, collettivamente indicate come *Post Object Programming* (POP); tra queste particolare successo sta riscuotendo negli ultimi cinque anni la programmazione orientata agli aspetti (*Aspect Oriented Programming*, AOP) che sta ricevendo un notevole impulso anche in ambito industriale grazie alla comparsa di tool open source affidabili ed accessibili che permettono di applicarne i principi anche all'interno del più tradizionale sviluppo orientato agli oggetti.
Questo articolo vuole essere un'introduzione alla programmazione orientata agli aspetti. Esso si divide in tre parti: la prima parte (Paragrafo 2) fornirà una breve presentazione

delle problematiche che hanno portato alla formulazione di questo nuovo paradigma, soffermandosi sulla nuova terminologia introdotta e sulle sue caratteristiche innovative; nella seconda parte (Paragrafo 3) si cercherà di capire quali siano le criticità che ancora limitano l'adozione di AOP nel processo di produzione di software in grande scala, e quali siano le necessità di evoluzione della ricerca e dell'industria in questo senso; infine, nella terza parte (Paragrafo 4) verranno delineati alcuni scenari concreti in cui è possibile applicare allo sviluppo del software, in modo efficace e produttivo, le idee della programmazione orientata agli aspetti.

2. PROGRAMMAZIONE ORIENTATA AGLI ASPETTI: UN'INTRODUZIONE

David Parnas, in un articolo del 1972 [4], sottolinea come una delle maggiori conquiste nell'ambito della programmazione sia stato lo sviluppo di tecniche, paradigmi e strumenti che permettono una chiara e netta modularizzazione del prodotto software. L'articolo in questione affronta quindi la tematica centrale di tutto il processo di progettazione, ovvero come sia possibile modularizzare il software in modo tale da poter garantire l'indipendenza tra i componenti e la loro completa riusabilità.
In termini più moderni, si usa definire come "*concern*" una funzionalità, un requisito che può essere pensato e sviluppato come entità autonoma; data questa definizione, il lavoro dell'architetto, sottolinea Parnas, consiste nell'individuare i concern che l'applicazione dovrà indirizzare e guidare il loro sviluppo cercando di mantenerne l'indipendenza: si parla quindi di *separation of concerns* (SoC) come uno degli obiettivi primari da raggiungere nel processo di sviluppo. A trent'anni di distanza la *separation of concerns* è ancora una tematica di grande attualità, e il fermento di ricerche e di proposte in questo ambito dimostra chiaramente che ancora non si è trovata una soluzione ottimale a questo problema.
Una delle caratteristiche che ha contribuito all'affermazione del paradigma di programmazione orientato agli oggetti è stata proprio la sua capacità di modellare come componenti autoconsistenti, dotati di chiaro scopo e responsabilità, le entità principali di business, dando origine a sistemi più modulari, riusabili e verificabili.
Tuttavia l'aumentare della complessità dei requisiti (e del software di conseguenza) ha mostrato, nella pratica, come non sempre la modularità riesce ad essere gestita e implementata così come sarebbe auspicabile sulla carta. Il problema centrale risiede nel fatto che in ogni componente responsabile di eseguire una certa funzionalità vengono ad interagire tipicamente molti concern; di questi, uno o pochi sono le competenze fondamentali per quell'ambito (dette funzionalità di business) mentre molti altri sono caratteristiche accessorie, dettate spesso dalla tecnologia stessa o da requisiti non funzionali.
Si faccia riferimento per esempio a quanto è rappresentato nella figura 1; il componente competente per "inserimento cliente nel database" si trova a dover avere a che fare con concern complementari, per esempio problemi di autorizzazione (l'utente è autorizzato ad inserire nuovi clienti?), di tracciatura, di politiche di connessione a database. Tutte competenze per cui il modulo stesso non è responsabile, e che sono utilizzate diffusamente in varie parti dell'applicazione. Ci si riferisce a queste funzionalità che toccano molti componenti di una applicazione con il nome di "*crosscutting concern*".
Ciò che si verifica è che la programmazione ad oggetti, con il suo modello basato sull'ereditarietà e sulle dipendenze, non riesce a modellare efficacemente i *crosscutting concern*. Infatti, se da una parte è vero che attra-

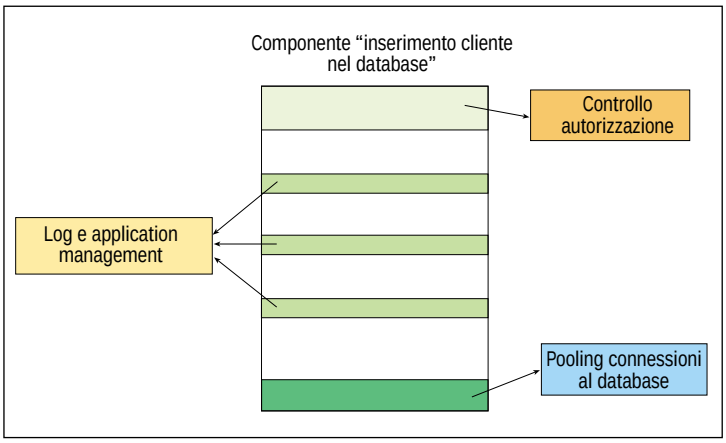


FIGURA 1
Compresenza di più concern all'interno di un singolo componente

verso questo paradigma è possibile disegnare moduli separati per la gestione di queste funzionalità di uso generale, è altrettanto vero che il meccanismo delle dipendenze fa sì che tutti gli altri moduli dell'applicazione siano intrinsecamente legati a questi. Questo fatto si manifesta chiaramente analizzando come nel codice che implementa una determinata logica di business si faccia esplicito e frequente riferimento a moduli esterni non direttamente legati a tale funzionalità.
Lo scopo con cui nasce la programmazione orientata agli aspetti è quella di fornire una base teorica, un linguaggio e degli strumenti per catturare e definire in modo sintetico i crosscutting concern e per permettere il loro utilizzo all'interno delle applicazioni minimizzando le dipendenze tra moduli diversi.

2.1. Terminologia della programmazione orientata agli aspetti

La programmazione orientata agli aspetti mette a disposizione due metodologie per indirizzare il problema dei crosscutting concerns, denominate rispettivamente *dynamic crosscutting e static crosscutting*.
Con *dynamic crosscutting* si intende la possibilità di aggiungere nuovi comportamenti ad un software esistente nel momento in cui questo viene eseguito. Questo significa fornire rispettivamente una codifica del comportamento aggiuntivo (il concern) e un set di punti dell'applicazione in cui esso deve essere eseguito.
In questo contesto, esattamente come la classe è il concetto fondamentale per la modellazione di una entità nella programmazione ad oggetti, un crosscutting concern è modellato completamente attraverso un *aspetto*.
L'aspetto si compone fondamentalmente di due parti:
- l'*advice*, che fornisce una completa implementazione del concern;
- il *pointcut*, che definisce una famiglia di punti nell'esecuzione del programma in cui l'advice deve essere eseguito

Il *pointcut* a sua volta è esprimibile come una combinazione di entità fondamentali, dette *join point*, che esprimono i punti caratteristici individuabili all'interno dell'esecuzione di un programma come l'invocazione di un metodo, di un costruttore, l'accesso ad un attributo o l'occorrenza di una eccezione.
L'utilizzo contemporaneo di una applicazione modellata ad oggetti (per tutte le funzionalità di business) e ad aspetti (per i crosscutting concern) rende il software molto più modulare e di semplice comprensione. Gli oggetti che compongono l'applicazione si occupano solamente della logica applicativa, senza fare alcun riferimento esplicito a problematiche di tipo tecnologico come quelle illustrate nella figura 1.
Ogni aspetto, d'altro canto, modella uno specifico crosscutting concern: al suo interno, l'advice specifica l'implementazione del crosscutting concern, mentre il pointcut tutto l'insieme dei punti dell'applicazione in cui esso va applicato. Sarà compito dell'infrastruttura (che sia essa la macchina virtuale Java, un framework specifico o il middleware) eseguire al momento opportuno gli aspetti all'interno del normale flusso di esecuzione dell'applicazione.
Come si può immediatamente intuire, l'utilizzo della tecnica dei pointcut si rivela uno strumento potente per migliorare la modularità di un'applicazione, perchè permette di esprimere sinteticamente all'interno dell'aspetto tutti i punti in cui dovranno essere aggiunte le funzionalità descritte dall'advice.
Ritornando all'esempio presentato in precedenza, la situazione si configura ora come illustrato nella figura 2. I tre concern non fondamentali del componente (controllo autorizzazione, log e pool di connessione al databa-

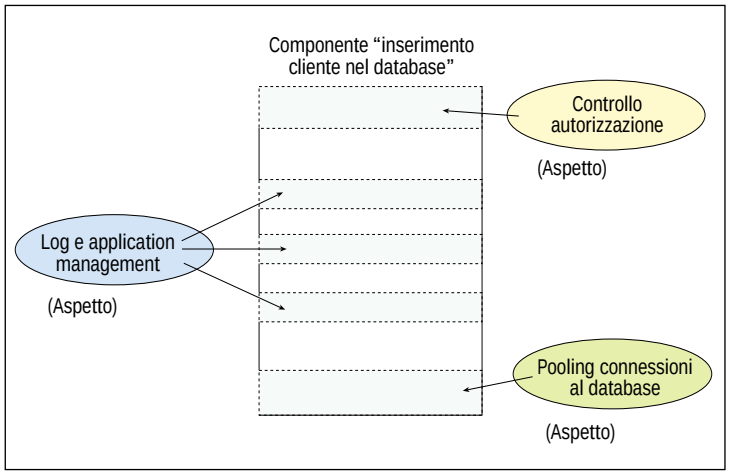


FIGURA 2
Applicazione degli aspetti all'interno di un componente

se) sono modellati come aspetti, e contengono al loro interno la definizione delle famiglie di punti del software in cui devono essere applicati. Ciò che si verifica è una inversione delle dipendenze: il componente "inserimento cliente nel database" ora non interagisce più esplicitamente con moduli esterni ma è influenzato da questi al momento dell'esecuzione. Questo significa altresì che nello sviluppo del componente non sarà più necessario occuparsi dei concern non fondamentali.
Lo *static crosscutting* si differenzia dal precedente perchè consente di alterare la struttura stessa di una famiglia di oggetti attraverso l'introduzione (si parla infatti di *introduction*) di nuovi attributi, metodi o interfacce. Anche in questo caso, l'alterazione della struttura determinata dai tool AOP non è visibile nel codice dell'applicazione, ma si manifesta solamente al momento dell'esecuzione.
Sebbene sia meno utilizzato e conosciuto rispetto al dynamic crosscutting, lo static crosscutting consente di snellire le gerarchie degli oggetti presenti nell'applicazione così come di aggirare le limitazioni di alcuni linguaggi, come Java, che non gestiscono ereditarietà multiple.

3. SFIDE E PROBLEMI NELL'ADOZIONE DELL'AOP NEL PROCESSO DI PRODUZIONE SOFTWARE

Pur rimanendo finora confinata alla popolazione dei ricercatori e dei pionieri delle nuove tecnologie, l'attenzione che sta montando attorno alla programmazione orientata agli aspetti è chiaramente misurabile considerando il numero sempre crescente di tool che nascono per permettere di utilizzare questo nuovo approccio con i linguaggi di programmazione più diffusi.
In particolare il linguaggio Java, grazie alla sua vasta base di sviluppatori, offre un'ampia e qualificata scelta in questo senso: dopo la nascita, nel 1996, del progetto precursore AspectJ [13], si è assistito negli ultimi anni al proliferare di tool analoghi, spesso finanziati e sponsorizzati dalle maggiori multinazionali del settore informatico. Tra questi particolare seguito hanno JBossAop [14], implementazione che è parte integrante dell'application server open source JBoss, e AspectWerkz [15], prodotto open source che gode della sponsorizzazione di BEA Systems.
Sul fatto che AOP possa un giorno essere adottato su grande scala le opinioni sono ancora constrastanti. Accade così che mentre IBM nella primavera di quest'anno dichiara che "l'Aspect Oriented Programming ha raggiunto livelli di affidabilità tali da permetterne un utilizzo commerciale"[6], James Gosling, uno dei creatori storici del linguaggio di programmazione Java, a soli pochi mesi di distanza osserva che "AOP è un po' troppo complicato per me, perchè promette bene nella teoria... ma il modo in cui viene messo in pratica tende ad essere troppo pericoloso" [7].
Non c'è dubbio che in queste dichiarazioni di verso opposto si nascondono i pregi e le problematiche di cui una futura adozione di AOP su scala industriale dovrà tenere conto. Da un lato, il nuovo paradigma di programmazione mette a disposizione nuove possibilità e strumenti per indirizzare in modo più efficiente problemi e temi rilevanti; d'altro canto rende necessario un ulteriore "paradigm shift" (cambio di paradigma) nelle competenze dei programmatori. E non c'è dubbio che quest'ultimo tema, già noto alle aziende che hanno dovuto riconvertire parte del loro personale dalle tecniche della programmazione strutturata a quelle della programmazione ad oggetti, rappresenti un'incognita che Gosling non manca di sottolineare.
Da questo punto di vista l'introduzione della programmazione orientata agli aspetti è agevolata dal fatto che questa si affianca e non sostituisce il paradigma più tradizionale ad oggetti, agendo quindi più come complemento che come avversario delle metolodogie di analisi, design e sviluppo attualmente in uso. Ciò rende possibile la creazione di tool che, consentendo di utilizzare congiuntamente entrambe le modalità di lavoro, possono rappresentare un acceleratore notevole per l'adozione di questo nuovo approccio.
Un ultimo aspetto che deve essere sottilineato è che la programmazione orientata agli aspetti manca tuttora di uno standard, che è un requisito forte per attirare gli investimenti delle aziende. Infatti il periodo a cui stiamo assistendo è ancora caratterizzato dalla nascita di nuovi progetti fortemente innovativi, che tentano di portare nuove idee e nuove

tecniche all'attenzione degli sviluppatori, e di conseguenza di guadagnare maggior popolarità, utilizzatori e finanziamenti. Si sente ancora la mancanza di un forte movimento atto a stabilire una linea comune in questo settore; su questa via si sta muovendo, a passi lenti, l'AOP Alliance [8], i cui standard sono ancora poco accolti e soprattutto non ancora molto sponsorizzati dai grandi gruppi multinazionali.

4. SCENARI DI ADOZIONE IN AMBITO INDUSTRIALE

Quando si parla di adozione in ambiente di produzione della programmazione orientata agli aspetti il primo esempio sempre citato è legato alle funzionalità di *logging* (tracciatura e registrazione di dati rilevanti durante l'esecuzione dell'applicazione). Si tratta infatti di un caso tipico di crosscutting concern, una operazione pervasiva che tocca tutti i moduli di una applicazione pur essendo un'aspetto marginale nel contesto di applicazioni complesse.
L'utilizzo di AOP per il logging rappresenta tuttavia solo un primo passo verso l'adozione di questa tecnologia, sicuramente necessario per cominciare ad conoscere ed approfondire i benefici che può portare, ma che non ne rivela appieno tutte le potenzialità.
Nella figura 3 è delineata una possibile evoluzione nell'utilizzo della programmazione orientata agli aspetti che, al crescere della maturità dei tool e della confidenza degli sviluppatori, può consentire di esplorarne appieno tutti i possibili utilizzi.
La curva di adozione ipotizzata nella figura 3 è composta di quattro fasi.
Nella prima, la programmazione orientata agli aspetti interviene come componente secondario di una applicazione già esistente, portando alcune nuove funzionalità interessanti ma non fondamentali, come il *logging* ed il *design by contract* (si veda il paragrafo 4.3. per una breve illustrazione di questa tecnica). In questa fase gli aspetti introdotti possono essere rimossi dall'applicazione, perchè non ne contribuiscono sostanzialmente alla funzionalità.
In una seconda fase, si assiste all'introduzione di aspetti ancora non fondamentali dal punto di vista funzionale, ma che comportano notevoli vantaggi all'applicazione. Rientrano in questa categoria tutte le tecniche per l'ottimizzazione delle prestazioni e la gestione dell'applicazione. La rimozione di questi aspetti non pregiudica il funzionamento dell'applicazione, ma possono essere avvertiti dall'utente finale (sotto forma di peggioramento dei tempi di risposta del prodotto) o dai gestori dell'applicazione.
Nella terza fase di adozione, AOP è utilizzata per modellare componenti responsabili di funzionalità applicative fondamentali, come la registrazione dei dati su database (persistenza) e l'applicazione di politiche di sicurezza e di autorizzazione. Tali aspetti fanno parte a tutti gli effetti della applicazione e non possono essere rimossi.
Infine, nella fase finale, si può ipotizzare che il paradigma di programmazione orientato agli aspetti diverrà parte integrante, assieme a quello orientato agli oggetti, del processo di progettazione del software. Ciò significa che già in fase di stesura del modello concettuale del software potranno essere identificati i principali concern ed assegnati rispettivamente ad oggetti e ad aspetti, secondo criteri formalizzati. Inoltre, la disponibilità di librerie di aspetti riutilizzabili potrà garantire un processo di sviluppo più snello ed efficiente.
Considerando ora il presente e l'immediato futuro, nella parte rimanente di questa sezione saranno delineati brevemente alcuni scenari di adozione tipici.

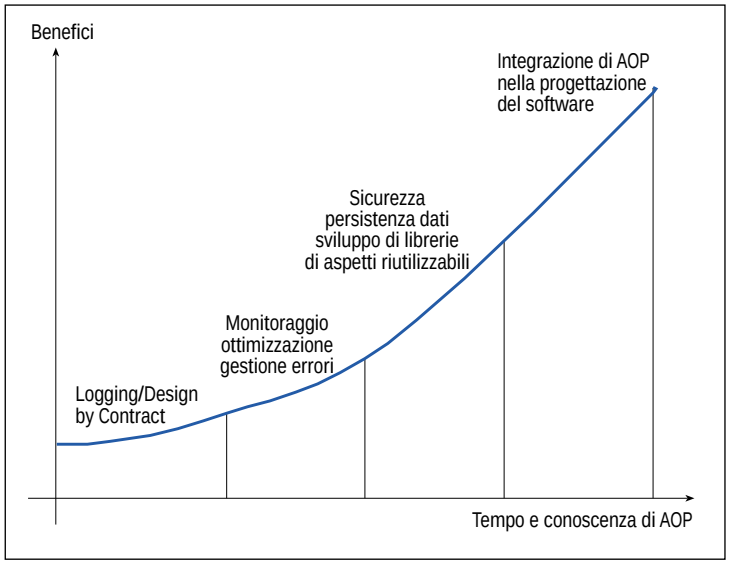


FIGURA 3
Curva di adozione di AOP

4.1. Monitoraggio dell'applicazione

Si tratta sostanzialmente di un'estensione e generalizzazione del logging che sfrutta la caratteristica della programmazione orientata agli aspetti di poter intercettare determinati eventi nel ciclo di esecuzione del software e di poter eseguire logiche opportune in queste circostanze.
Nel caso specifico è possibile in modo immediato valutare i tempi di esecuzione di determinate procedure [9] e il loro numero di invocazioni o effettuare un controllo sul numero di istanze di oggetti create. Rispetto ad un approccio più tradizionale AOP consente di variare in modo flessibile la definizione di quali parti dell'applicazione sono soggette a monitoring e quali controlli effettuare senza per questo richiedere alcuna modifica del codice applicativo oggetto dell'analisi.

4.2. Caching e ottimizzazione delle prestazioni

L'ottimizzazione delle prestazioni è una tematica spesso critica nello sviluppo di applicazioni aziendali, a causa di due fattori concomitanti: in primo luogo, l'esigenza di introdurre miglioramenti nelle performance emerge spesso nelle fasi avanzate di sviluppo, se non perfino ad applicazione già in produzione, a causa del manifestarsi di problemi di carico o di eccessivi tempi di risposta; inoltre, le tecniche utilizzate in questo ambito sono spesso complesse, soggette ad errori e di conseguenza oggetto di continui miglioramenti ed aggiustamenti.
L'utilizzo della programmazione orientata agli aspetti ha in questa ottica il grande pregio di permettere di disaccoppiare nettamente il codice responsabile della logica applicativa da quello relativo alla ottimizzazione delle prestazioni , consentendo di applicare senza problemi tali tecniche anche a posteriori, direttamente in ambiente di produzione e senza modificare il codice applicativo.

4.3. Design by Contract

I problemi di integrazione tra applicazioni diverse spesso portano al manifestarsi di errori inaspettati la cui causa è difficile da determinare proprio per la stretta interazione tra moduli differenti, spesso sviluppati da diversi fornitori, a volte anche costruiti con tecnologie differenti. In questo contesto risulta di grande utilità l'adozione di una tecnica nota come *Design by Contract* [10] che permette di verificare la correttezza dell'interfacciamento tra moduli diversi andando a controllare la conformità a determinati requisiti (detti *contratti*) dei dati di ingresso e di uscita da un'applicazione.
La programmazione orientata agli aspetti consente di applicare il controllo di aderenza a specificati contratti tra applicazioni in modo flessibile [11]; è possibile porre sotto controllo ogni singolo metodo di un'applicazione, e definire politiche di monitoraggio diverse a seconda dell'ambiente sotto osservazione (sviluppo, test, collaudo, produzione).

4.4. Gestione della sicurezza

La tematica della sicurezza è anch'essa centrale nello sviluppo di applicazioni aziendali; anche in questo caso la programmazione orientata agli oggetti consente di progettare un modulo indipendente per la gestione della sicurezza, ma è normalmente compito degli sviluppatori interfacciarsi correttamente ad esso nello scrivere le funzionalità di business.
Grazie alla programmazione orientata agli aspetti è possibile mettere in atto una più netta *separation of concerns*, facendo in modo che un esperto di sicurezza sviluppi indipendentemente un modulo dedicato, dichiarando esplicitamente (all'interno di specifici *aspetti* di sicurezza) quali regole vadano applicate ed in quali punti dell'applicazione; grazie a questo approccio, gli sviluppatori delle funzionalità di business sono in grado scrivere il loro codice senza mai trattare esplicitamente le problematiche di sicurezza.
L'applicazione di AOP consente alla sicurezza non solo di verificare i diritti di un certo profilo utente ad eseguire determinati servizi, metodi, costruttori, ma anche di gestire correttamente il verificarsi di errori critici, applicare algoritmi di crittografia a dati sensibili oppure segnalare tentativi di accesso anomali al sistema [12].

5. CONCLUSIONI

La programmazione orientata agli aspetti rappresenta una grossa opportunità per imprimere un'ulteriore evoluzione agli strumenti ed alle modalità con cui viene sviluppato il software. Il suo maggiore punto di forza consiste nel fatto che è possibile introdurre questo nuovo paradigma a fianco del tradizionale processo di sviluppo in maniera graduale, cominciando ad apprezzarne i benefici senza sostenere investimenti iniziali eccessivi. Per sfruttare appieno le sue caratteristiche innovative è comunque necessario investire in un cambio di paradigma che consentirà di integrare meglio queste nuove teorie nel processo di progettazione del software, in modo da poter identificare già in fase di analisi quale sia l'approccio più efficace per indirizzare ogni *concern* relativo ad una applicazione.
Non mancano tuttavia le incognite che ne possono rallentare l'adozione: la mancanza di uno standard unitario innanzitutto e di conseguenza anche il non perfetto supporto di queste nuove tecnologie all'interno delle piattaforme software (application server, middleware) sviluppate dalle più importanti società.

Bibliografia

[1] Succi Giancarlo: L'evoluzione dei linguaggi di programmazione: analisi e prospettive. *Mondo Digitale*, Anno II, n. 8, Dicembre 2003, p. 39-52.

[2] Kiczales Gregor, Lamping John, Mendhekar Anurag, Maeda Chris, Lopes Cristina, Loingtier Jean-Marc, Irwin John: *Aspect Oriented Programming*. Atti della European Conference on Object-Oriented Programming (ECOOP), Finland. Springer-Verlag LNCS 1241, 1997.

[3] Elrad Tzilla, Filman Robert E., Bader Atef: Aspect Oriented Programming. *Communication of the ACM*, Vol. 44, Ottobre 2001.

[4] Parnas David L.: On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, December 1972

[5] Laddad Ramnivas: *AspectJ in Action – Practical Aspect Oriented Programming*. Manning Publication Co., 2003

[6] *IBM to make aspect-oriented development a reality*. CNET News.com, http://news.com.com/2100-1008_3-5178164.html?tag=nefd_top

[7] *Sun's Gosling: New Java Flavors Brewing*. eWeek, http://www.eweek.com/article2/0,1759,1624844,00.asp

[8] *AOP Alliance*, http://aopalliance.sourceforge.net/

[9] Krishnamurthy *Ramchandar: Performance Analysis of J2ee Applications Using AOP Techniques*. ONJava.com, http://www.onjava.com/pub/a/onjava/2004/05/12/aop.html

[10] Meyer Bertrand: *Object Oriented Software Construction*. Prentice Hall, 1997.

[11] Diotalevi Filippo: *Contract enforcement with AOP*. DeveloperWorks, http://www-106.ibm.com/developerworks/java/library/j-ceaop/

[12] Viega John, Bloch J.T., Chandra Pravir: Applying Aspect-Oriented Programming to Security. *Cutter IT Journal*, Vol.14, n. 2, p. 31-39, 2001.

[13] Home page del progetto AspectJ, http://www.aspectj.org

[14] JBossAop, un tool alternativo ad AspectJ, http://www.jboss.org/developers/project/jboss/aop

[15] AspectWerkz, un tool alternativo ad AspectJ, http://aspectwerkz.codehaus.org

FILIPPO DIOTALEVI lavora come IT Specialist presso IBM Italia a Milano, occupandosi di tematiche di progettazione e sviluppo di applicazioni su architettura J2ee. È autore di articoli tecnici e divulgativi di argomento informatico apparsi su riviste e siti italiani ed internazionali.
filippo.diotalevi@it.ibm.com