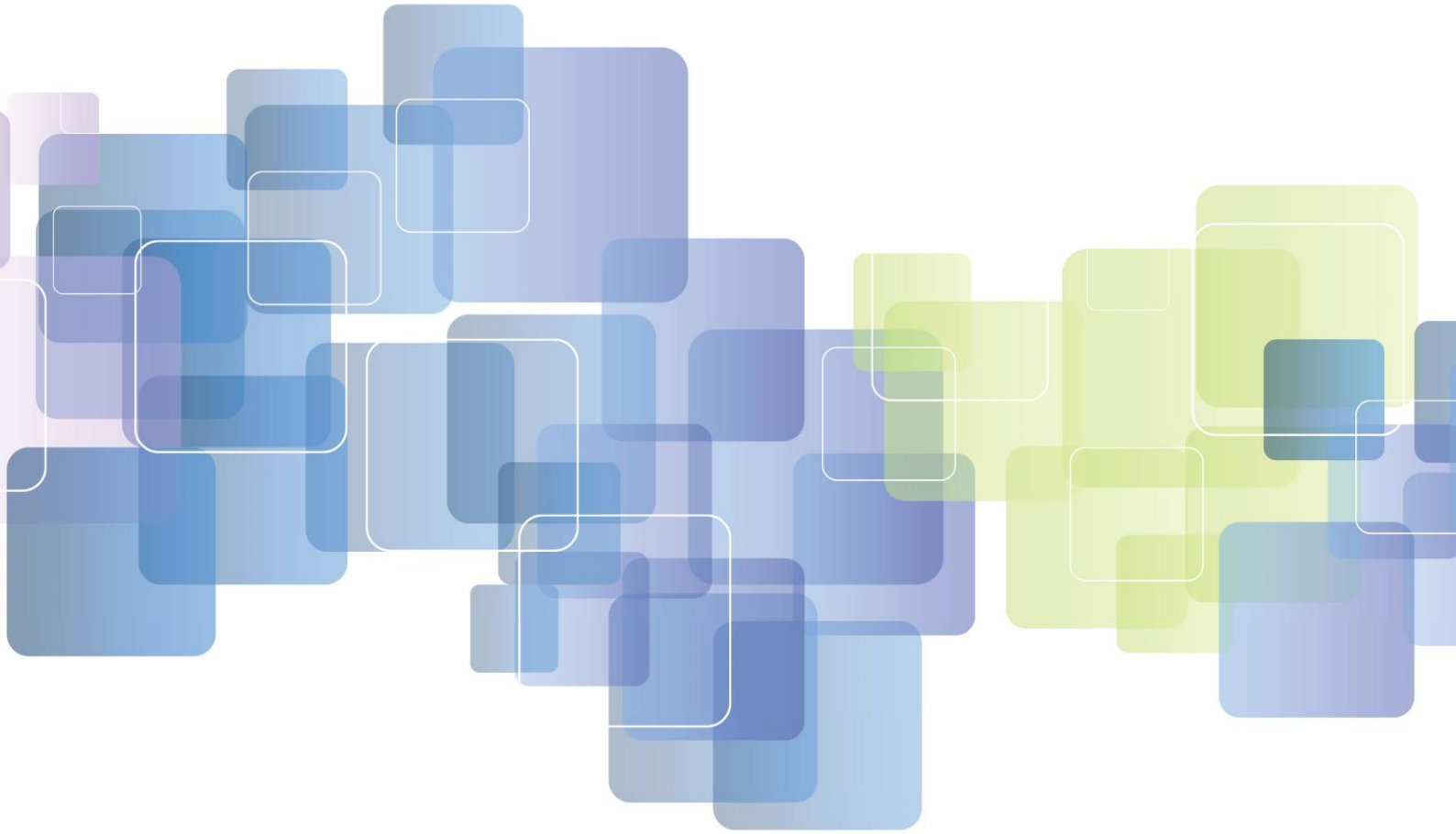
TecnetDati

Approccio OO versus approccio strutturato

Iolanda Salinari



L'approccio OO promette di colmare i limiti dei metodi tradizionali

Nel momento in cui si fa sempre più evidente la necessità di un approccio formale all'analisi e al disegno di sistemi Internet / Intranet, ci si chiede quali siano i metodi e le tecniche più adatte per applicazioni di questo tipo.
Sebbene il mercato indichi chiaramente che l'orientamento a oggetti diventerà gradualmente l'approccio preferito per tutti i tipi di applicazione e non solo per quelle Internet / Intranet, l'approccio strutturato è tuttora molto diffuso, in particolare sono ancora ampiamente utilizzati modelli dell'analisi strutturata, quali l'ERD per la rappresentazione dei dati.
Per quanto concerne specificatamente le applicazioni Internet/Intranet diventa sempre più comune l'impiego di tecniche di analisi e di disegno OO, o quantomeno di concetti OO dove non si utilizza ancora alcun approccio formale.
D'altra parte esistono ottime ragioni per raccomandare l'OO:

- il "caso d'uso" , che rappresenta oggi un concetto cardine dell'analisi e dell'intero ciclo di vita OO, è un modo "naturale" per modellare le interazioni tra utente e sistema (a testimonianza di ciò l'interesse che questo concetto suscita in modo via via crescente nella comunità informatica)
- l'approccio OO enfatizza il riuso tramite concetti quali l'ereditarietà, i componenti (su quest'onda si è sviluppata un'intera industria che si basa sullo sviluppo per componenti)
- se l'implementazione prevede linguaggi OO (es. Java), è sensato che anche l'analisi e il disegno che precedono la realizzazione siano effettuati in ottica OO, per evitare inutili e costose "conversioni".

L'esperienza degli ultimi anni ha rilevato una serie di limiti nei metodi tradizionali, che sono diventati tutti argomenti a favore dell'approccio OO e delle sue promesse di superare queste carenze.

Approccio all'analisi meno naturale

L'approccio strutturato distingue nettamente tra dati e funzioni, l'OO supera questa dicotomia con il concetto chiave di Oggetto.
I teorici dell'OO sostengono che modellare la realtà sulla base di questo concetto è molto più naturale "Non si può concepire un processo . . . che non coinvolga anche dei dati. In breve, dati e operazioni sui dati appaiono così strettamente legati nella nostra mente, che occorre cogliere elementi degli uni e degli altri per mettere a fuoco un qualsiasi concetto che ci aiuti a capire i processi d'elaborazione".

Assenza di concetti quali incapsulazione e information hiding

La netta bipartizione tra modelli dei dati e modelli delle funzioni non consente di applicare in modo rigoroso i principi dell'incapsulazione e dell'information hiding e di trarne in modo opportuno i conseguenti benefici.
Incapsulazione e information hiding consentono un'organizzazione più efficace del software fin dalle fasi precoci del ciclo di vita (già in analisi).
Infatti l'analisi e il disegno OO si sviluppano intorno al concetto di oggetto, che per definizione

- è il frutto dell'incapsulazione in un'unica struttura di dati e funzioni
- consiste di una porzione pubblica, interfaccia o protocollo di accesso e di una porzione privata in cui sono incapsulati i dati e i metodi (operazioni o servizi) che operano su tali dati.

Ogni classe di oggetti è un mondo autonomo, che comunica con le altre classi esclusivamente mediante messaggi (i messaggi ovvero le possibili richieste di servizio costituiscono la porzione pubblica di un oggetto).
L'applicazione di concetti quali l'incapsulazione e l'information hiding fornisce innegabili vantaggi:

- una riduzione della complessità perché per usare un oggetto non è necessario conoscerne la struttura interna, è possibile utilizzarlo indipendentemente dalla sua implementazione
- una forma di "indipendenza logica dei dati" permettendo modifiche all'implementazione degli oggetti senza obbligare a modifiche anche delle applicazioni che li usano.

Sicuramente questi concetti non sono nuovi, hanno una stretta parentela con i principi di modularità che perseguono alta coesione e basso coupling, principi propri del disegno strutturato, ma nell'approccio OO, a differenza di quanto avveniva nei metodi tradizionali, si fondano sul concetto cardine di oggetto, possono essere applicati in modo più rigoroso e fin dalla fase di analisi.
L'esempio che segue mostra la differenza tra i due approcci, strutturato e OO, nell'analisi di una particolare funzionalità.
A scopi puramente illustrativi, in entrambi i casi viene utilizzata la notazione del DFD.

Inserimento di un ordine del cliente - approccio strutturato tradizionale

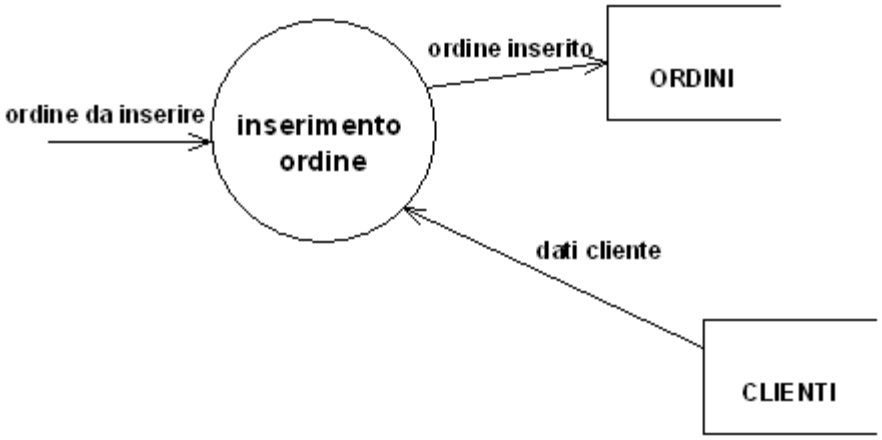


figura 1

Il processo *inserimento_ordine* accede al data store CLIENTI per verificare l'esistenza del cliente prima di memorizzare l'ordine nel data store ORDINI.

Inserimento di un ordine del cliente - approccio OO

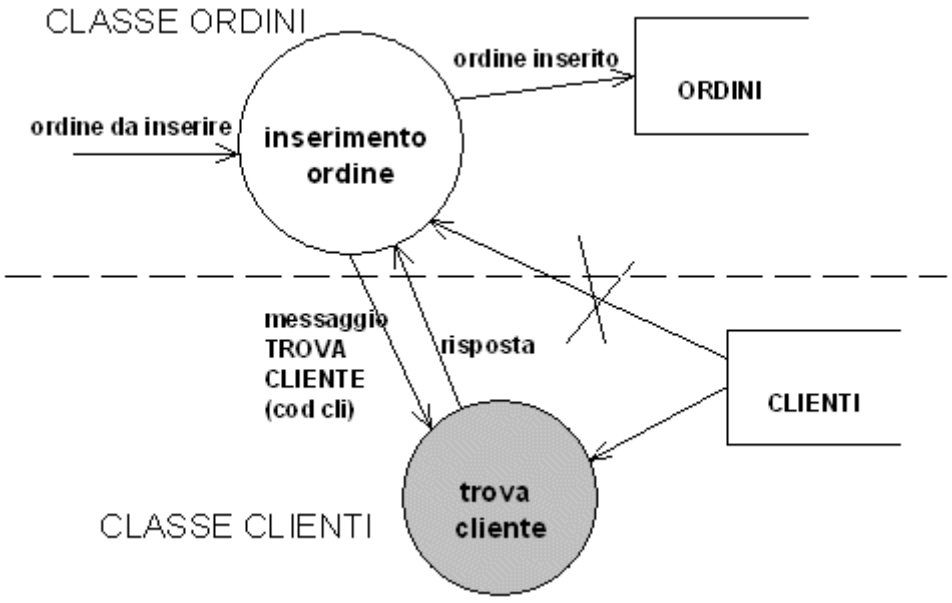


figura 2

L'oggetto ORDINE, che ha la responsabilità di memorizzare l'ordine (operazione *inserimento_ordine*), per portare a termine il suo compito richiede una collaborazione all'oggetto CLIENTE per verificare l'esistenza del cliente dell'ordine, tramite l'invio del messaggio *trova_cliente*. Questo messaggio innesca, nell'oggetto CLIENTE, l'operazione corrispondente *trova_cliente*, che fornisce una risposta all'oggetto ORDINE, ecc.. Sebbene in entrambi i casi sia stata usata la medesima notazione (il DFD), è chiaro che la prospettiva cambia completamente in ottica OO: sempre per i principi di incapsulazione e information hiding, non esistono processi (metodi o operazioni) che

possano accedere ai dati di più classi di oggetti. Se un oggetto ha bisogno dei dati di un oggetto appartenente ad un'altra classe, poiché non può accedervi direttamente, deve richiedere una collaborazione all'oggetto in questione. La comunicazione tra oggetti diversi avviene mediante messaggi.

Difficoltà nel passaggio dall'analisi al disegno

L'approccio strutturato prevede modelli diversi per l'analisi e il disegno. E' oggettivamente difficile passare da un modello come il DFD a quello della carta strutturata, i teorici non sono mai arrivati ad un accordo preciso su come attuare la transizione dall'analisi al disegno.
Per contro nel disegno OO non si mettono da parte i modelli prodotti in analisi, ma gli stessi modelli sono riutilizzati e portati ad un maggior livello di dettaglio (ad es. un diagramma di sequenza o un diagramma delle classi prodotti in analisi saranno affinati e perfezionati in fase di disegno).

Poca attenzione alle interazioni utente sistema

L'analisi strutturata non dà particolare rilievo agli aspetti riguardanti l'interazione tra utenti e sistema, che diventano un fattore sempre più critico ed importante per le applicazioni attuali.
Per contro il modello dei casi d'uso dell'OO si basa principalmente sulla determinazione di questi aspetti: il caso d'uso è per definizione la descrizione dell'interazione utente-utilizzatore/sistema.

Minore attenzione alla distribuibilità delle applicazioni

L'approccio strutturato nasce per un ambiente target monopiattaforma, di conseguenza non prevede fondamenti teorici o elementi formali per la distribuibilità delle applicazioni. Al contrario l'OO si adatta molto bene al paradigma client/server, in particolare per quanto riguarda:

- la definizione delle mappe (interfacce GUI)
- lo scambio di comunicazione tra piattaforme diverse è quello proposto dall'OO, avviene tramite messaggi
- l'organizzazione del software, così come si delinea già in fase di analisi, è particolarmente adatta alla distribuzione, perché l'elemento di distribuzione è la classe di oggetti, che per definizione è un insieme coeso di dati e funzioni.

Inoltre il disegno OO, secondo gli standard UML, prevede uno specifico modello per rappresentare la distribuzione delle componenti, il diagramma di allocazione (deployment diagram), che evidenzia la configurazione dei nodi elaborativi in ambiente di run-time, con i componenti, processi e oggetti ubicati in questi nodi.

Scarsa riusabilità del codice

Sicuramente l'esigenza e i tentativi per attuare la riusabilità non sono qualcosa di nuovo, già alla fine degli anni '60, in occasione di una conferenza organizzata dalla Nato, venivano coniati termini quali "ingegneria del software" e "software factory" e si prediva con tono quasi profetico una produzione di massa del software a partire da componenti riusabili.

Purtroppo nei trent'anni che sono seguiti la riusabilità ha fatto ben pochi passi avanti, l'approccio top-down, strutturato, non ne ha certo promosso l'applicazione. Tuttavia, con la progressiva affermazione del paradigma OO, la marcia verso la riusabilità ha segnato una brusca accelerazione, su quest'onda è nata negli ultimi anni un'intera industria che si basa sul principio di "assemblare il software a partire da componenti prefabbricate".

L'object oriented consente di applicare i principi della riusabilità fin nelle prime fasi di sviluppo del software, infatti i casi d'uso prevedono meccanismi di estensione e astrazione che preludono alla riusabilità dei casi d'uso stessi (vedi relazioni tra casi d'uso <> e <>).

Questi concetti sono strettamente imparentati con quelli analoghi di specializzazione / generalizzazione e di aggregazione tra le classi di oggetti; l'ereditarietà è il meccanismo che permette di diffondere porzioni di codice all'interno del sistema.

Limiti della scomposizione top-dow

L'approccio top-down dell'analisi strutturata si basa sul processo di scomposizione funzionale: il DFD può essere partizionato su più livelli, permettendo di passare gradualmente dalla vista globale dell'intero sistema (diagramma di contesto) ad una vista più dettagliata ad ogni livello di scomposizione, fino ad individuare i cosiddetti processi elementari. Se da un lato questo modo di procedere consente di scomporre un problema complesso seguendo una disciplina di pensiero logica e ben organizzata, dall'altro presenta limiti evidenti.

I criteri su cui basare la scomposizione del sistema in sottosistemi non sono univoci (si parla di criteri strutturali, per eventi, per ciclo di vita, ecc..). Sebbene nella scomposizione al primo livello sia prevalso generalmente il criterio per eventi, resta la difficoltà di scomporre i livelli intermedi di un DFD. Non essendo questi criteri univoci, spesso l'analista "itera più volte" modellando il DFD, aggregando più bolle a creare un livello superiore, scomponendo ulteriormente una bolla, spostando una bolla sotto un altro "padre", per un obiettivo essenzialmente "estetico", per una maggiore chiarezza e leggibilità del diagramma.

Purtoppo queste attività possono essere qualche volta piuttosto onerose rispetto ai benefici apportati, non dimentichiamo che l'obiettivo fondamentale consiste nell'individuazione dei processi elementari e del comportamento nel loro insieme. Quando poi il sistema cambia, diventa un vero disastro, l'aggiunta di un nuovo processo può provocare la revisione di più livelli del DFD, gli analisti devono affrontare la prospettiva di una riprogettazione continua.

Con questo tipo di approccio è sicuramente più difficile incrementare il sistema con nuove funzionalità, mentre l'approccio OO, in quanto basato sui casi d'uso, facilita uno sviluppo incrementale.

Considerazioni finali

Quanto detto finora riguarda essenzialmente limiti riferiti agli aspetti sostanziali, di contenuto, dell'approccio strutturato, ma non esclude a priori l'utilizzo di elementi formali dei modelli previsti da questo approccio, almeno per quanto riguarda la fase di analisi.
Se, ad esempio, allo stato attuale gli analisti in azienda hanno un maggiore familiarità con le tecniche del DFD o dell'ERD, è possibile utilizzare ancora questi formalismi, ma è necessario calarli in un contesto metodologico diverso, in un'ottica OO, per consentire l'applicazione di principi importanti quali l'incapsulazione, l'information hiding, l'ereditarietà e il polimorfismo, per evitare i limiti dell'analisi strutturata tradizionale.
Per quanto riguarda l'ERD questo adattamento è più semplice, il modello E/R viene a rappresentare la componente dati del modello delle classi OO.
Il modello delle classi è in realtà un'estensione dell'Entity Relationship: ha in aggiunta meccanismi di astrazione quali la specializzazione/generalizzazione (IS_A) e la composizione/aggregazione (IS_PART_OF) e ovviamente i metodi (operazioni).
Per quanto riguarda il DFD, questo potrebbe essere usato non più per scomporre l'intero sistema per passi successivi, ma piuttosto per rappresentare il comportamento di un singolo caso d'uso.

Ovviamente in questo modo viene ad esprimere significati che non sono necessariamente quelli dell'analisi strutturata tradizionale. In particolare:

- gli agenti esterni corrispondono agli attori del sistema
- i processi alle operazioni degli oggetti
- i data flow agli eventi o messaggi
- i data store alla componente dati dell'oggetto.

Per un esempio vedi figura 2, che illustra il comportamento parziale di un caso d'uso secondo queste convenzioni.

A questo proposito si ricorda che i formalismi dei DFD hanno inizialmente trovato accoglienza in metodi per l'analisi OO come quelli di Booch, Martin & Odell e Rumbaugh, ma con la progressiva affermazione di UML sono stati tuttavia generalmente "sconfessati" e sostituiti da altre rappresentazioni come i Diagrammi di Interazione (diagrammi di sequenza e di collaborazione) e di Attività.

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