# Integrazione dei modelli concettuali e logici nei sistemi e nella loro rappresentazione

Piero Magnani

## *Premessa*

E’ frequente la convinzione che il contenuto delle fasi di modellazione dei dati di un’applicazione informatica debba comprendere solo le entità e relative correlazioni che interessano l’area che essa tratta.

Questo è vero nei casi, per la verità poco frequenti, di applicazioni che hanno una vita autonoma, senza collegamenti con altre applicazioni. In queste condizioni, tutti i dati necessari sono contenuti nell’applicazione stessa e l’applicazione interagisce con l’esterno solo con il dialogo verso l’utente finale.

Penso si possa concordare che nell’ambito aziendale le applicazioni di qualche rilievo interagiscono sempre salvo casi in cui, per ragioni più politiche che tecniche, si vuole mantenere l’isolamento. Ma anche nel campo di applicazioni industriali di processo, diventano sempre più frequenti i casi in cui le funzioni applicative devono integrarsi.

Nelle applicazioni aziendali l’integrazione delle funzionalità si è imposta ben presto. Per esempio, da tempo l’applicazione di gestione delle spedizioni è strettamente collegata a monte con quella degli ordini , a valle con quella della fatturazione e della contabilità ed è completamente connessa all’applicazione anagrafica che viene sempre più generalizzata per la gestione di tutte le informazioni sui soggetti che hanno contatti con l’Azienda.

Il problema che ci poniamo in questo scritto è *se* e *come* rappresentare queste integrazioni nel modello dati, a qualsiasi livello: concettuale, logico e fisico. Tratteremo le applicazioni aziendali, ma quanto detto vale anche per quelle di processo.

## *Proprietà dei dati*

E’ un concetto del quale si è parlato tanto a lungo da rischiare di perderne il significato. Si afferma che il dato è un patrimonio aziendale, ma dal punto di vista della sua gestione troviamo queste situazioni:

1. *tutti aggiornano tutto e tutti leggono tutto*
2. *solo alcuni aggiornano e tutti leggono tutto*
3. *solo alcuni aggiornano e solo alcuni leggono*

### Tutti aggiornano tutto e tutti leggono tutto

In questo caso non esiste nessuna regola e, salvo vincoli dovuti alla riservatezza, ogni programma, modulo o utente può operare liberamente sui dati; è evidente che in queste condizioni non vi è nessuna nozione di proprietà dei dati. I dati sono a disposizione di tutti per qualsiasi operazione di lettura o di scrittura sia richiesta.

Ne derivano due problemi:

- l’impossibilità di attribuire a qualcuno la responsabilità dei valori del dato e
- la difficoltà di individuare quali oggetti saranno interessati a un cambiamento formale o semantico o di dominio del dato.

Il primo problema può essere affrontato con l’uso di meccanismi di accesso, in particolare DBMS, che mantengano traccia delle operazioni di archivio; purtroppo la lettura di questi log è sempre molto difficile e il riconoscimento dell’oggetto responsabile risulta sempre insicuro.

Per il secondo problema esistono analizzatori di codice e monitor di accessi. Anche qui l’uso è difficile. Per esempio è quasi impossibile risalire al chiamante di un modulo, quando la chiamata è stata fatta attraverso il valore di una variabile (chiamata dinamica).

Di conseguenza la realizzazione di una qualsiasi modifica è molto pesante e incerta

#### Solo alcuni aggiornano e tutti leggono tutto

In questo secondo caso si accetta che solo alcuni programmi, moduli o utenti possono aggiornare, mentre tutti possono leggere i dati direttamente.

Per capire la situazione è importante stabilire se la capacità di aggiornamento sono attribuite con criteri aderenti al confine fra applicazioni, per cui solo una o poche applicazioni possono modificare il dato. Solo così possiamo dire che esiste un inizio, anche se timido, di impiego del concetto di proprietà del dato.

In questa ipotesi tutte le applicazioni che intendono modificare il dato devono rivolgere una richiesta alla proprietaria del dato. Ovviamente si è nella condizione migliore quando la proprietaria è unica, ma si potrebbe anche poter gestire la situazione di molte proprietarie, quando queste sono fortemente collegate e hanno un solo responsabile.

Rimane comunque la difficoltà di individuare le conseguenze di una modifica delle caratteristiche formali o sostanziali del dato e di identificare gli elementi che richiedono una conseguente manutenzione.

#### Solo alcuni aggiornano e leggono

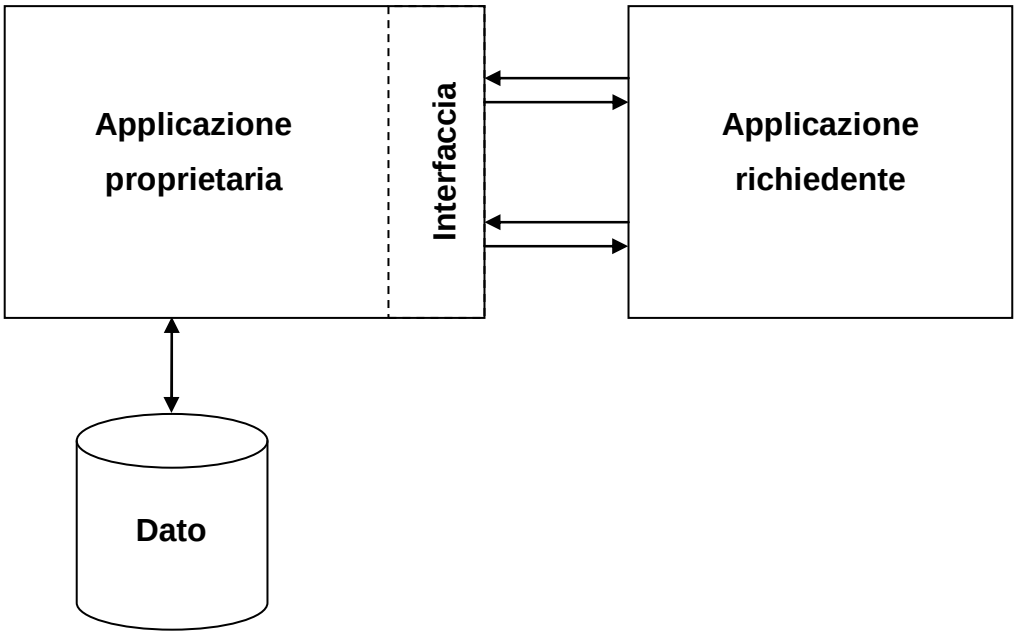
Qui siamo nella situazione ideale dove i dati sono completamente proprietà di un'applicazione (per semplicità ipotizziamo che sia unica e la chiameremo *proprietaria*). Tutti gli oggetti non appartenenti ad altre applicazioni (che chiameremo *richiedenti*) devono chiedere alla proprietaria sia l'ottenimento dei dati, che la loro modifica perché solo quella ha la capacità di accesso.

Con questo il principio della proprietà del dato è solidamente stabilito e la gestione degli errori e della variazione delle caratteristiche è notevolmente agevolata. Infatti, se la struttura di comunicazione funziona correttamente è sufficiente condurre la ricerca nell'ambito di una sola applicazione con gli strumenti che le sono propri e che sono conosciuti da una ben determinata cerchia di specialisti,

## *Interfacce*

L'interfaccia è l'insieme dei moduli che permettono il colloquio fra due applicazioni. Il tipo di interfaccia che trattiamo è quella che permette a un'applicazione di richiedere l'accesso ai dati di un'altra applicazione.

Per rispettare il principio della proprietà dei dati l'interfaccia deve essere formata da moduli appartenenti all'applicazione proprietaria. Lo schema che ne risulta è quello a cui faremo riferimento come *Soluzione base*

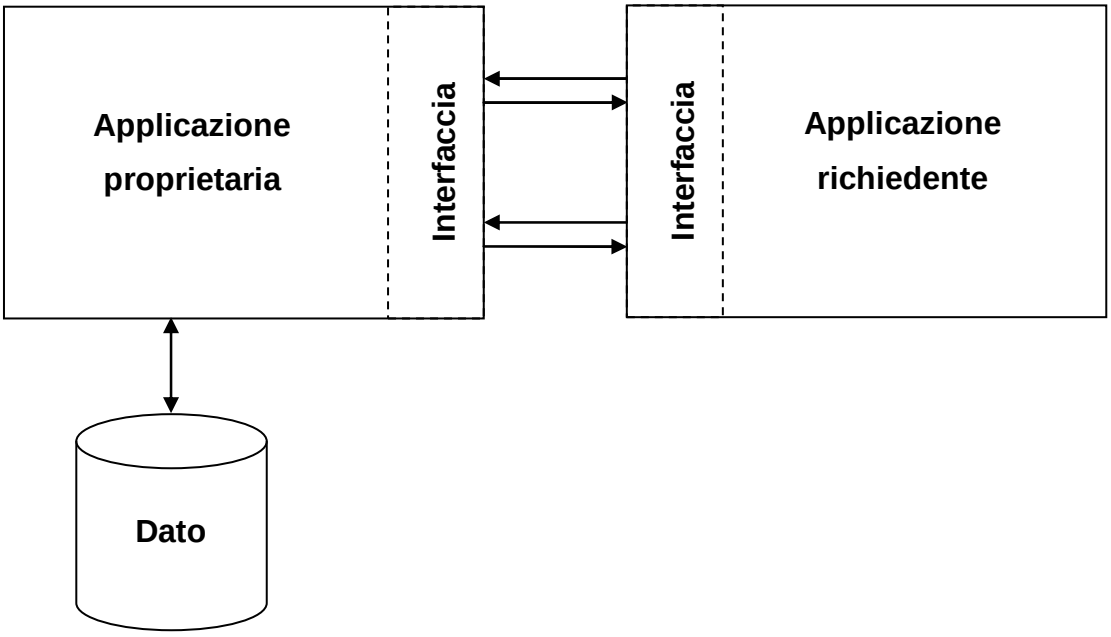



Questa soluzione presenta lo svantaggio che gli oggetti che realizzano le funzioni dell'applicazione richiedente percepiscono i dati come sono offerti dall'applicazione proprietaria attraverso la sua interfaccia.

Questo può comportare due rischi:

- quello di non vedere i dati nella forma ottimale richiesta dall'applicazione richiedente e
- quello di essere esposti a modifiche nelle modalità di trasmissione, nel formato e contenuto dei dati, con la conseguente necessità di adattare gli oggetti dell'applicazione richiedente tramite interventi di manutenzione.

Si può mitigare questo problema con uno strato di interfaccia anche nell'applicazione richiedente. Chiamiamo lo schema risultante *Soluzione bilaterale*

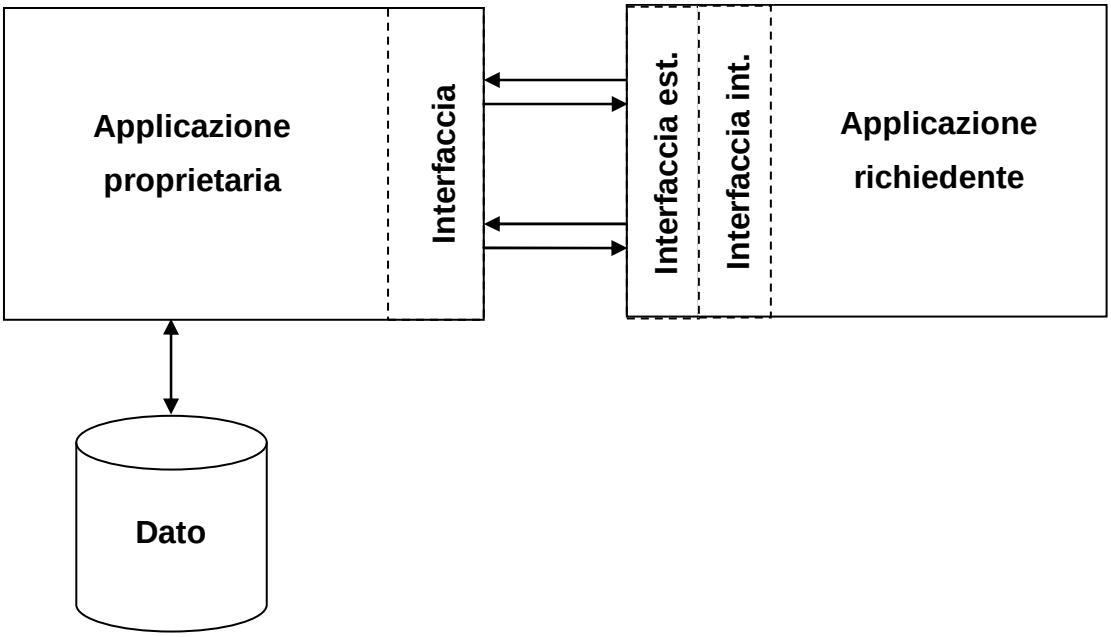


Con questa soluzione l'interfaccia dell'applicazione richiedente trasforma i dati inviati e ricevuti dall'interfaccia dell'applicazione proprietaria per adattarli agli oggetti dell'applicazione richiedente.

Il beneficio di questa soluzione si manifesta quando più applicazioni richiedenti devono accedere ai dati di quella proprietaria, per cui i servizi e il formato dei dati trasmessi non possono che essere un compromesso fra le esigenze delle varie applicazioni richiedenti o, addirittura, prescindere da queste esigenze perché al momento della costruzione dell'interfaccia le applicazioni richiedenti non sono ancora conosciute o approfondite.

L'interfaccia propria di ciascuna applicazione richiedente permette di ottimizzare l'uso dei dati da parte degli oggetti relativi. Non solo, ma in caso di modifica dell'interfaccia dell'applicazione proprietaria la manutenzione è necessaria solo nei moduli delle interfacce delle applicazioni richiedenti e non tocca i moduli funzionali, a meno di modifiche che lo richiedano espressamente per motivi aziendali. Comunque, anche in questa situazione la presenza di moduli di interfaccia nell'applicazione richiedente permette la diluizione degli interventi nel tempo e l'adattamento alle caratteristiche delle singole applicazioni.

E' possibile inserire nella struttura un ulteriore strato di interfaccia. Lo schema seguente delinea una *Soluzione a più livelli* mostrando questo nuovo livello sull'applicazione richiedente, ma la stessa tecnica può essere usata anche per l'applicazione proprietaria.




Nell'applicazione richiedente con l'interfaccia esterna si utilizzano i servizi offerti dall'applicazione proprietaria e si restituiscono dati relativamente semplici e, comunque, congruenti con quelli visti da quei servizi. Con l'interfaccia interna è possibile raggruppare ulteriormente i dati e presentare insiemi che possono essere direttamente utilizzati, evitando anche la duplicazione di operazioni di aggregazione e migliorando il riutilizzo del codice.

Va detto che questo strato di servizi potrebbe essere considerato parte integrante delle funzioni dell'applicazione, piuttosto che uno strato di interfaccia. La distinzione, peraltro, non avrebbe conseguenze pratiche, quindi, non approfondiremo ulteriormente questo discorso.

## *Modello concettuale*

Con queste premesse possiamo affrontare il nostro argomento principale.

Il modello concettuale dei dati è strettamente legato alla percezione che l'Azienda, attraverso i diversi tipi di utente, da quelli direzionali a quelli operativi, ha della realtà in cui si trova a operare e dei dati che la descrivono permettendo l'interazione e l'uso del sistema informativo.

L'entità corrisponde agli oggetti che l'Azienda riconosce nell'attività sottoposta ad automazione. Quando si sostiene che essa è l'elemento fondamentale del modello si afferma il principio base che presiede alla formazione di questo modello. Gli attributi descrivono le entità e le correlazioni stabiliscono quali sono i collegamenti fra loro; quindi tutto ruota intorno al concetto di entità.

Il modello concettuale serve come strumento di razionalizzazione delle informazioni che vengono via via raccolte e come mezzo per facilitare la corretta comunicazione all'interno e all'esterno del team di lavoro.

Qui non vogliamo entrare nel merito delle diverse scuole metodologiche per la formazione del modello. Prendiamo solo atto che esistono entità, attributi e relazioni, per capire se e come possiamo rappresentare i dati che non sono propri dell'applicazione.

### Integrazione del modello concettuale

E' possibile, almeno teoricamente, raggruppare tutti i modelli dei dati delle applicazioni interconnesse in un unico modello. Purtroppo questo diventerebbe troppo grande da gestire e di difficile lettura, non solo per la quantità di informazioni contenute, ma anche per la necessità che il lettore sia capace di interpretare correttamente i dati delle varie applicazioni.

Possiamo, quindi, escludere questa ipotesi anche quando il numero delle applicazioni sia limitato. Rimangono due alternative:

1. inserire nel modello dati solo le entità proprie dell'applicazione,
2. aggiungere al modello le entità delle altre applicazioni.

Nel primo approccio la descrizione delle entità estranee è lasciata alla documentazione dell'interfaccia delle applicazioni proprietarie e a documenti più o meno informali. E' un modo di lavorare che può avere il vantaggio della semplicità (più apparente che reale) ma la costruzione dei moduli delle applicazioni riceventi richiede l'esame di più documenti per individuare l'intera serie dei dati che interessano e risulta piuttosto complesso.

Nel secondo approccio le entità delle altre applicazioni saranno descritte nel modello come devono essere viste dagli oggetti di quella sotto studio. Quest'ultima affermazione è particolarmente importante alla luce delle caratteristiche e dell'utilizzo del modello concettuale, come sono state presentate nel precedente paragrafo. Infatti, le entità devono essere descritte come percepite dagli utenti di questa applicazione, facendo, naturalmente, riferimento e armonizzando questa visione alla percezione degli altri utenti. A questo livello non ci sono considerazioni tecnologiche.

Inoltre, nelle prime fasi del progetto, è possibile che l'appartenenza delle entità alla singola applicazione possa non essere ancora precisata e una visione allargata dei dati coinvolti è capace da una parte di permettere di proseguire il ciclo di sviluppo, dall'altra di aiutare a stabilire le appartenenze.

Il secondo approccio ha, quindi, l'indubbio vantaggio di riunire in un unico documento le informazioni necessarie, ma rischia di allargare il modello in modo pericoloso.

L'unica regola che si può proporre consiste nel mantenere una stretta aderenza all'analisi funzionale per identificare gli elementi (entità, attributi, correlazioni) che sono strettamente necessari. Purtroppo nulla può sostituire la capacità di sintesi e un sano buon senso nel progettista, al quale non bastano, anche per altri aspetti, le conoscenze dei metodi e degli strumenti. Infatti, tutti i dati aziendali sono correlati e se si seguono i legami senza spirito critico si finisce con il fare uno schema onnicomprensivo e inutilizzabile.

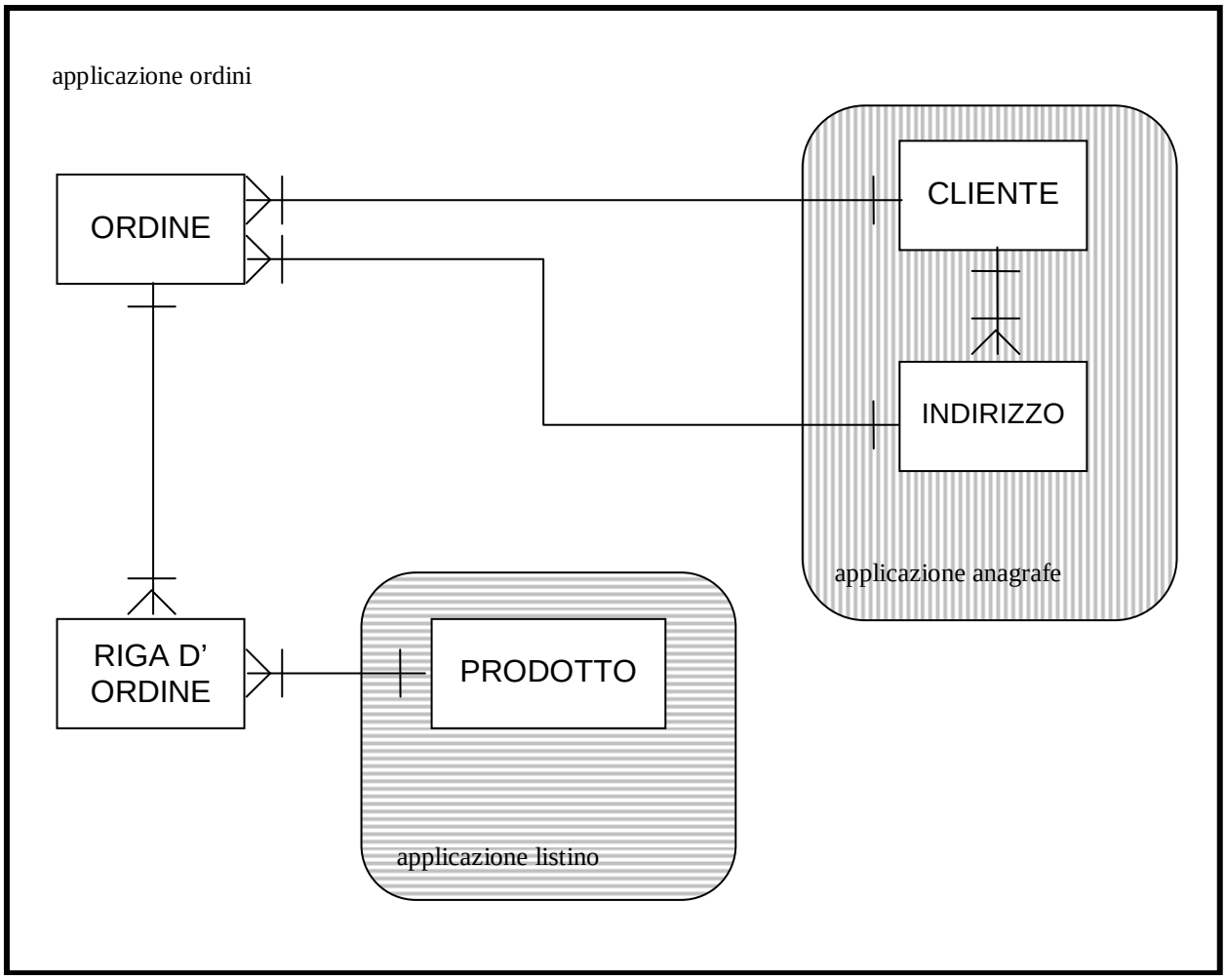
Nella realtà se consideriamo la natura del modello concettuale, che, come abbiamo detto sopra, è la rappresentazione della visione aziendale e non informatica, possiamo accettare alcune semplificazioni. Per fare un esempio nell'applicazione Anagrafe l'entità Soggetto può avere una quantità molto ampia di entità correlate e dare luogo a uno schema molto complesso. Nell'applicazione Ordini è percepita o una sola entità Soggetto oppure quella stessa entità a cui si aggiunge l'Indirizzo perché potrebbe essere necessario specificare, per esempio, separatamente l'indirizzo di fatturazione e quello di consegna.

### Rappresentazione del modello concettuale integrato

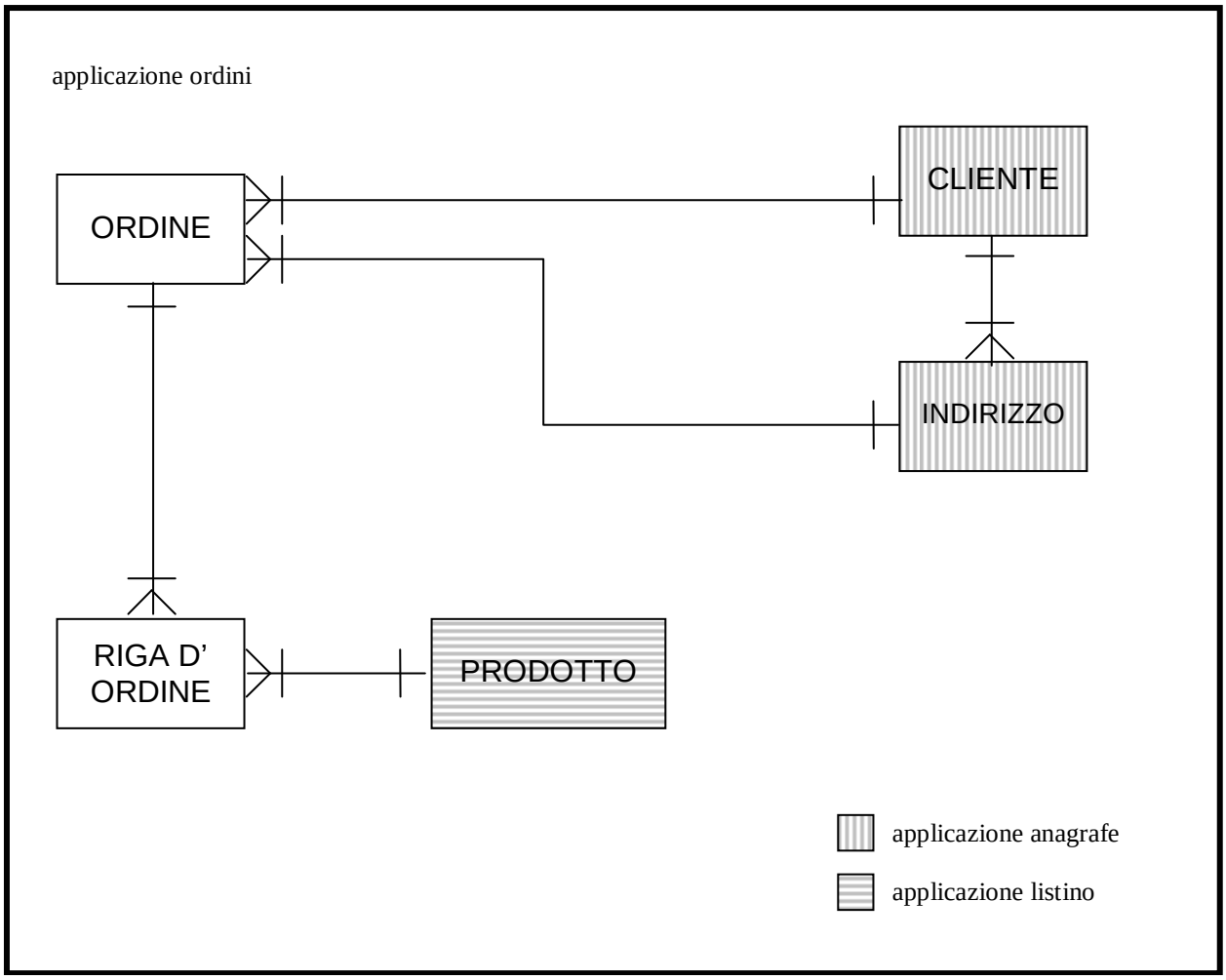
La rappresentazione grafica del modello concettuale che si considera è quella Entity/Relationship allargata agli attributi, ricordando che questi ultimi non sempre sono necessari in tutte le fasi della discussione.

Vi sono due modi per distinguere graficamente le entità proprie dell'applicazione da quelle esterne, appartenenti ad altre applicazioni:

– l’utilizzo di riquadri, opportunamente commentati, che racchiudono le entità esterne; ogni riquadro comprende entità di una determinata applicazione;



– l’utilizzo di un codice di colori per cui il simbolo di entità ha un colore diverso, o nella riga esterna o nel riempimento, in funzione dell’applicazione di appartenenza.




Nella lettura degli schemi si noti che delle due relazioni fra ordine e anagrafe una ha il significato di “fattura a” e si appoggia al Cliente perché l’indirizzo di fatturazione è una sua caratteristica generale, mentre l’altra rappresenta “spedisci a” ed è specifica del singolo ordine. E’ anche evidente che per l’utente di questa applicazione l’entità in questione è Cliente, anche se nell’anagrafe essa potrebbe essere trattata come Soggetto o Controparte.

I due metodi sono equivalenti, ma nel modello concettuale è, forse, preferibile il primo perché indica con maggiore chiarezza le applicazioni con cui quella sotto studio si deve integrare.

Se consideriamo il modello concettuale come un documento di lavoro che gradualmente si completa con l’allargarsi delle conoscenze sulle funzioni, è evidente che l’evidenza dei riquadri aiuterà la crescita della comprensione dell’insieme delle applicazioni.

Per minimizzare il numero dei componenti dello schema è necessario evitare l’inserimento di elementi non necessari come, ad esempio, entità di dominio. D’altra parte si dovranno aggiungere tutti e soli quegli attributi che l’applicazione deve trattare per l’esecuzione delle funzioni. Questo modo di procedere aiuterà nel momento della progettazione delle interfacce.

Durante la costruzione del modello concettuale può capitare di non essere certi sull’appartenenza dell’entità a un’applicazione. Questa situazione di incertezza può essere rappresentata nel primo

modo con il riquadro che interseca il rettangolo dell'entità, oppure con un particolare riquadro che racchiude le entità in dubbio. Nel secondo modo si può usare un colore specifico oppure un tratteggio particolare.

## *Modello logico*

Il modello logico è il modello di lavoro nel momento in cui si progetta l'aspetto tecnico dell'applicazione. In questo modello entità, correlazioni e attributi sono in funzione da una parte delle necessità degli oggetti elaborativi, dall'altra delle caratteristiche del modello di Database adottato. Conseguentemente il modello logico è il risultato e la guida dell'analisi tecnica. Questa, infatti, deve produrre il modello logico, che, successivamente, deve essere diligentemente rispettato nella descrizione degli elementi architetturali che verranno successivamente realizzati.

### Integrazione del modello logico

Per l'integrazione di dati provenienti da altre applicazioni nel modello logico ritroviamo le due alternative che abbiamo considerato per il modello concettuale:

- lasciare la descrizione dei dati esterni alla documentazione delle interfacce,
- comprendere nel modello logico le entità esterne.

Una parola sulle interfacce: troppo frequentemente le interfacce sono considerate una parte secondaria, che può essere differita nel tempo. In questo modo tutte i progetti che si fanno sugli utilizzatori (moduli e programmi) di questi dati esterni si basano solo su ipotesi di come saranno le interfacce.

Risulterebbe sufficiente, a questo punto, fissare la vista delle interfacce, cioè stabilire quali sono i servizi che devono rendere e i dati che richiedono nei due sensi. Purtroppo può capitare che l'ipotesi di vista dell'interfaccia si riveli non realizzabile o attuabile solo con un impegno eccessivo. Da qui la necessità di rivedere in parte il disegno tecnico o affrontare un maggior costo per la costruzione delle interfacce.

A questo si aggiunga che l'inizio della fase di test richiede la presenza dell'interfaccia. In mancanza, anche questo è frequente, si devono scrivere dei moduli di simulazione dell'interfaccia, quasi sempre con l'accesso a tabelle che devono essere definite e popolate appositamente. Alla disponibilità dell'interfaccia le prove dovranno essere tutte rifatte perché nessuno può garantire che il funzionamento sia perfettamente uguale a quello dei moduli di simulazione.

D'altra parte le interfacce sono moduli. Chi deve progettare questi moduli deve avere a disposizione un modello dati, anzi, ne deve avere a disposizione due, almeno per i dati che devono transitare:

- il modello logico dell'applicazione proprietaria dei dati e
- quello dell'applicazione richiedente.

Supponendo che l'interfaccia sia costruita, secondo lo Soluzione base, sull'applicazione proprietaria in modo generalizzato, comune a tutte le applicazioni che richiederanno quei dati, il secondo modello è ancora necessario e consiste nella vista logica che la prima vuole fornire alle altre.

Avendo a disposizione questa vista logica è utile inserire nel modello logico dell'applicazione richiedente gli elementi che saranno effettivamente utilizzati. Una verifica assicurerà che questi elementi corrispondano a quelli definiti nel modello concettuale o che comunque questi possano essere derivati con ragionevole impegno nello sviluppo.

Un diverso approccio consiste nell'inserire nel modello dell'applicazione richiedente gli elementi esterni così come le funzioni dell'applicazione pretendono. Questo comporta la collaborazione con il proprietario dei dati per stabilire se i dati originari possono alimentare quella vista. Successivamente potrebbe essere realizzata un'interfaccia opportuna dall'applicazione proprietaria

(Soluzione base) con l'eventuale aggiunta di un'interfaccia sull'applicazione richiedente (Soluzione bilaterale).

In ogni caso la disponibilità degli elementi esterni sul modello logico permette una più immediata comprensione dei dati e facilita il progetto.

#### Rappresentazione del modello logico integrato

Anche le possibilità di rappresentazione grafica sono ancora le stesse esaminate per il modello concettuale: uso di riquadri o uso di colori diversi.

Nel modello logico si dovrebbe preferire la seconda soluzione perché

- un'entità esterna può derivare dalla combinazione di dati di due applicazioni e questo renderebbe difficile l'uso dei riquadri che si verrebbero a intersecare;
- entità esterne derivanti dalla stessa applicazione potrebbero collegarsi con entità proprietarie in punti diversi nel modello; pertanto il secondo tipo di rappresentazione lascia più libertà nella costruzione del grafico.

## *Modello fisico*

Evidentemente i dati esterni non devono comparire nel modello fisico. In termini Erwin saranno, quindi, logical only.

## *Sviluppi futuri*

Nella descrizione del modello logico è stato sempre dato per certo che in quella fase la divisione dei dati fra le applicazioni possa essere conosciuta o determinata.

In effetti ci sono molti casi in cui i dubbi possono permanere:

- quando una nuova applicazione è prevista, ma non ancora pianificata;
- quando la pianificazione di un'altra applicazione non ne permette la disponibilità in tempo per la partenza di quella considerata;
- quando l'applicazione proprietaria non è ancora ben conosciuta, o nel suo contenuto, o nell'uso che l'utenza vorrà farne;
- quando per problemi di prestazione o di parallelismo operativo si preferisce duplicare dei dati.

In tutti questi casi nel modello logico compaiono delle entità che saranno poi realizzate come tabelle fisiche e i cui dati devono essere compresi fra quelli proprietari.

Se vogliamo lasciare un sufficiente grado di elasticità nel nostro sistema abbiamo un'opportunità: quando l'entità sia in qualche modo in dubbio consideriamo l'accesso a questa entità come una particolare interfaccia, realizzata attraverso uno o più moduli.

Quando si presentasse l'evento che richiede lo spostamento dei dati in un'altra applicazione, saremmo sempre in grado di sostituire l'accesso fisico alla tabella con una richiesta all'interfaccia dell'altra applicazione, modificando i moduli di interfaccia della nostra applicazione, ma senza toccare le funzioni principali.

Dal punto di vista grafico le entità trattate in questo modo dovrebbero essere riconoscibili con un colore o un tratteggio diversi.

## *Conclusioni*

Nell'articolo si è discussa la fattibilità e l'opportunità di allargare gli schemi prodotti durante le fasi di disegno concettuale e logico dei dati con elementi appartenenti ad applicazioni diverse da quella considerata.

L'allargamento è reso realizzabile dalla constatazione che il numero degli elementi introdotti può essere contenuto a un livello accettabile attraverso la scelta di quelli che sono strettamente necessari per la realizzazione degli obbiettivi funzionali.

Così facendo si ottiene una visione generale dei dati con un unico documento, favorendo non solo il progetto dell'applicazione, ma anche quello delle interfacce che la collegano con altre applicazioni.

La visione dei dati che ne risulta è personalizzata alle necessità dell'applicazione sotto studio e rende più comprensibile l'insieme dei dati esterni, costituendo uno strumento fondamentale per lo sviluppo applicativo e un favorevole e solido punto di partenza per la progettazione e la programmazione delle interfacce, per le quali non sarà mai raccomandato a sufficienza un'attenta considerazione nella pianificazione e controllo del progetto.

Le considerazioni sull'integrazione hanno portato ad accennare a possibili alternative per l'architettura delle interfacce. Ciascuna delle alternative proposte rispetta il principio della proprietà del dato e fornisce uno schema per la strutturazione tecnica delle interfacce.

I metodi per realizzare l'integrazione nel modello concettuale e in quello logico non si differenziano se non per come sono inseriti nella natura della fase. Solo la loro forma grafica, per i motivi che sono state presentati, si diversifica proprio per adattarsi a quella natura. Nella fase di disegno concettuale si darà maggiore importanza al raggruppamento per applicazione proprietaria, mentre nella fase di disegno logico prevarrà la vista delle singole entità nella posizione grafica più consona per i collegamenti che si instaurano fra le entità interne o esterne all'applicazione.

I requisiti richiesti al sistema di supporto al disegno si limitano alla possibilità di disegnare riquadri per le varie aree, differenziare colori e tratteggi e, soprattutto, rendere possibile la definizione di elementi che saranno presenti fino al modello logico senza essere trasferiti nel modello fisico. Il prodotto Erwin ha tutte queste capacità.

Quindi, questa integrazione degli elementi di altre applicazioni nel disegno dati non solo è possibile, ma anche del tutto vantaggiosa. Il suo costo è relativamente basso e controbilanciato da benefici molto importanti per la qualità del sistema applicativo in costruzione e per la sua mantenibilità nel tempo.