

Target Cross

The screenshot displays the Target Cross ERP software interface. The main window is titled 'Archivio = SQL Server Azienda = 3 - Four Bytes'. The interface is divided into several panels:

- Ordini Clienti:** A form for managing customer orders, showing fields for Cliente (000203), Causale (DC), Data (10.06.2002), and various status indicators.
- Diagramma di Gantt:** A Gantt chart showing work cycles for 'Assemblaggio' and 'Fresatura' across different work centers (BO.01 to BO.06) from Saturday 14.09.02 to Sunday 15.09.02.
- Ciclo di lavoro 2002-ORP-0000007,0001:** A detailed view of a work cycle, showing machine time (412.00), operator time (412.00), and start/end dates (14.09.2002 to 16.09.2002).
- Totale per categorie:** A 3D bar chart titled 'Vendite comparate' showing sales data for 'Importi in Euro' across 'Categorie prodotti'. The chart compares 'Vendite prec.' (previous sales) and 'Vendite periodo' (sales in the period). The chart is attributed to 'Four solutions srl'.
- Carico macchine:** A bar chart showing machine load percentages for various work centers (BO.05, BO.06) across different dates (Mer 13.09, Gio 14.09, Ven 15.09, Sab 16.09, Dom 17.09, Lun 18.09, Mar 19.09, Mer 20.09, Gio 21.0).

The interface also includes a menu bar with options like 'Archivio', 'Modifica', 'Agende', 'Tabelle', 'Anagrafiche', 'Contabilità', 'Magazzino', 'Acquisti', 'Vendite', 'Produzione', 'Statistiche', and 'Manutenzione'. A toolbar at the bottom shows icons for 'Totale per categorie', 'Diagramma di Gantt', 'Ciclo di lavoro', and 'Ordini Clienti'.

Top technology

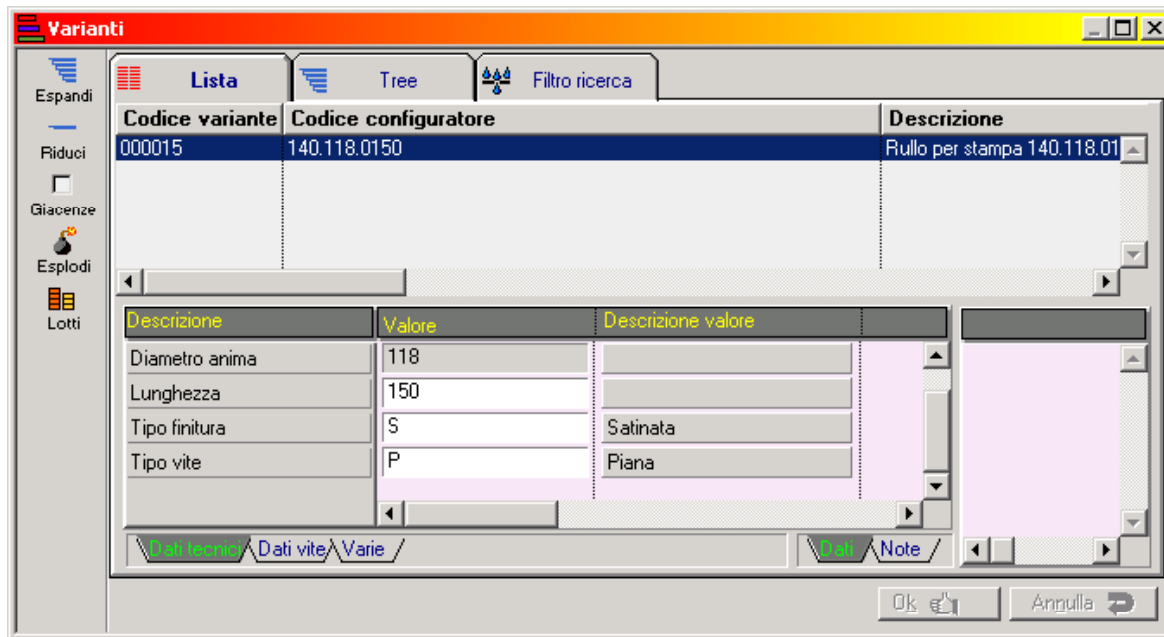
Le esclusive possibilità di personalizzazione permettono a **Target Cross** di adattarsi alla realtà operativa in cui si troverà ad operare meglio di qualunque altro software.

I suoi esclusivi meccanismi di personalizzazione gestionale permettono una maggiore velocità e facilità operativa, e consentono di evitare i rischi ed i problemi generalmente connessi alla customizzazione degli ERP.

La personalizzazione di Target Cross

La caratteristica più innovativa di Target Cross è la sua capacità di adattamento alle esigenze dell'Azienda, utilizzando una serie di motori di personalizzazione che permettono di adattare la struttura dati, le visualizzazioni di input – output, le stampe, la logica di funzionamento, il modo di calcolo alla realtà in cui dovrà operare.

Prima di esaminare queste caratteristiche, vale la pena esaminare il problema dell'adattabilità del software, e le soluzioni proposte fino ad oggi. Solo così sarà possibile capire perché e come la soluzione Target Cross rappresenti un vero e proprio salto generazionale.



Il problema della personalizzazione del software.

Tutti i programmi, indipendentemente dai contenuti, necessitano di essere adattati alle varie realtà in cui si troveranno ad operare. Non è concepibile un sistema che prenda in esame tutte le possibili variabili e peculiarità di tutti i settori merceologici in cui si troverà ad operare.

Qualunque prodotto gestionale non può quindi prescindere dalla necessità di consentire al System Integrator la possibilità di modifiche o aggiunte.

Questa necessità, a parole, viene recepita dalla quasi totalità dei prodotti in commercio, ad esclusione di alcuni prodotti di fascia decisamente bassa.

In pratica ci si accorge che per questo problema una soddisfacente soluzione non è certo facile da reperire sul mercato..

I limiti dei sistemi attualmente in commercio.

I metodi di personalizzazione attualmente sul mercato si possono dividere in alcune categorie. Spesso all'interno dello stesso programma si hanno combinazioni di più di una di esse.

Personalizzazione del sorgente

E' il sistema più drastico, molto praticato. Si prende un programma standard e se ne crea una copia personalizzata, che con il passare del tempo si distacca sempre più dall'originale.

Apparentemente non comporta controindicazioni e non ne avrebbe **se il software fosse un prodotto statico**. Date le continue evoluzioni, ad ogni versione si presenta il dilemma di cosa fare: **rifare le personalizzazioni o rinunciare alla nuova versione?**

Inoltre presuppone una conoscenza del programma quasi pari a quella di chi l'ha scritto.

Fornitura del sistema di sviluppo

Variante del punto precedente. Come apparente facilitazione allo sviluppo, viene fornito anche un macrolinguaggio per lo sviluppo delle modifiche e delle personalizzazioni. Valgono le stesse obiezioni di cui sopra. Inoltre, **questi sistemi di sviluppo “caserecci” sono creati, almeno per quanto riguarda i fornitori italiani, con budget e risorse molto ristretti, per cui non sono assolutamente paragonabili** a quelli forniti da società specializzate, che lavorano sul mercato mondiale e i cui costi sono ammortizzati su un numero di Utenti enormemente più elevato.

Personalizzazione a moduli

Variante della personalizzazione del sorgente: si cerca di individuare le aree funzionali da personalizzare e si modificano i **sorgenti solo dei moduli interessati**. Da gli stessi problemi della soluzione precedente, solo più localizzati. In pratica le funzioni aggiunte sostituiscono quelle standard, e si ha un programma ibrido.

Verticalizzazione

Può essere definita come una **“personalizzazione collettiva”**. In pratica spesso nasce come una personalizzazione, che viene venduta ad altri Utenti. Rispetto a questa, presenta il vantaggio di rimanere sotto il controllo di chi l’ha sviluppata, ed in teoria dovrebbe essere sempre aggiornata. In pratica, la scarsa diffusione di ogni versione personalizzata, che si rivolge ad una ristretta platea di Utenti, fa sì che le verticalizzazioni non tengano quasi mai il passo con il programma standard. In non pochi casi vediamo addirittura che la mole di verticalizzazioni tarpa le ali al programma standard, che blocca il proprio sviluppo proprio per non perdere la compatibilità con i suoi “dialetti”.

Parametrizzazione

Con questo termine si intende una metodologia che trasferisce **fuori dal codice oggetto vero e proprio alcuni parametri**. Ad esempio, se vogliamo avere un programma multilingua, possiamo inserire le scritte in lingua in un file esterno, che può essere modificato.

Ottima soluzione, ma estremamente limitata; non può ad esempio intervenire pesantemente sul funzionamento delle procedure. In pratica viene utilizzata solo per personalizzare le schermate di input - output.

Campi “jolly”

Nella struttura dati si creano dei **campi non utilizzati**, che di volta in volta vengono usati per gli scopi richiesti. Si intuisce che il numero di problemi risolvibili in questo modo è assai limitato.

Personalizzazione delle stampe

Nel campo dei reports si è assistito allo sviluppo di strumenti di personalizzazione assai validi, tanto che oggi in molti programmi questo tipo di modifiche non sono un problema. Peccato che **non siano estensibili all’intero software**.

La tecnologia “ad oggetti” o “a componenti”

Ultimamente la panacea per questi problemi è parsa essere la tecnologia ad oggetti.

Dato che il programma è costituito da oggetti, basterà **modificare gli oggetti desiderati e riportare queste modifiche nelle versione successive**.

Pur costituendo un deciso passo avanti rispetto alla personalizzazione dei sorgenti, **questo approccio non è esente da inconvenienti, che subdolamente non sono evidenti fin dall’inizio**.

Immaginiamo di volere personalizzare la scheda cliente del nostro programma, inserendo un nuovo campo che abbiamo aggiunto alla struttura dati.

Per fare questo creeremo una versione custom della finestra (l’oggetto) che rappresenta la nostra scheda Cliente. Questa sarà memorizzata da qualche parte.

Nelle versioni successive del programma, riporteremo la nostra finestra custom nel programma standard.

Supponiamo però che nel programma standard venga aggiunto qualcosa che a noi interessa, proprio in questa finestra. A questo punto avremo la scelta tra perdere questo miglioramento, o ricreare la modifiche alla finestra. Di solito, infatti, il processo di miglioramento di un programma coinvolge la maggior parte dei componenti.

Inoltre, non esiste nessuna garanzia che una finestra della versione ad esempio 2.0 funzionerà nella versione 3.0.

La situazione in pratica è gestibile se gli oggetti customizzati sono pochi, altrimenti diventa un problema.

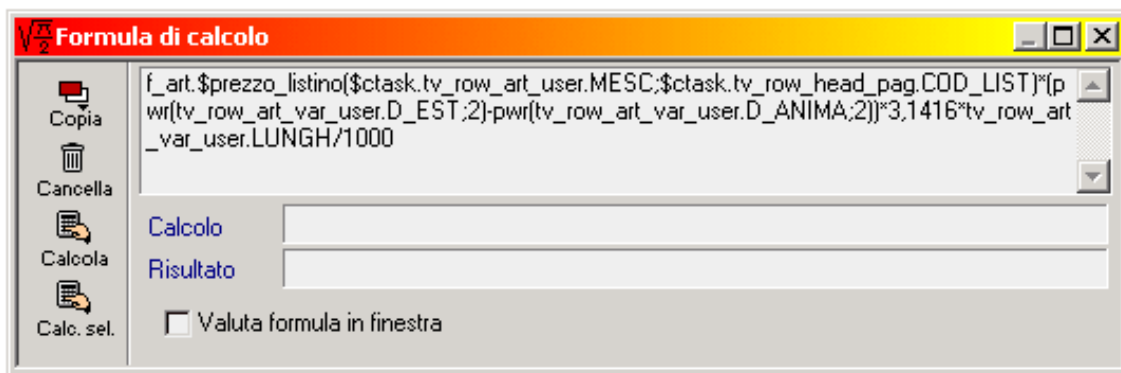
La soluzione Target Cross

Dall'esame di quanto esposto si evidenzia che il problema delle personalizzazioni dei programmi è a tutt'oggi ben lontano dall'essere risolto.

Target Cross ha l'ambizioso obiettivo di offrire una soluzione ai problemi di personalizzazione e adattamento che sia un vero salto qualitativo rispetto a quanto visto fino ad oggi; il suo scopo è permettere all'Utente o al rivenditore di apportare al programma modifiche funzionali di entità rilevante, senza costi di mantenimento successivi e senza perdere la compatibilità con il programma standard.

I componenti della soluzione Target Cross

La tecnologia Spreadsheet di gestione a formule



Pur non facendo parte a stretto rigore dei meccanismi di gestione delle modifiche, la gestione a formule gioca un ruolo di primo piano nel motore di personalizzazioni.

La personalizzazione migliore è quella che non c'è

La gestione a formule di Target Cross riduce drasticamente il numero e l'entità delle personalizzazioni da apportare. In moltissimi casi, situazioni che tradizionalmente dovevano fare ricorso ad una personalizzazione possono venire risolte utilizzando questa esclusiva funzione.

L'anello di congiunzione

La formula costituisce l'anello di congiunzione tra la personalizzazione apportata ed il programma standard, permettendo di influenzarne il funzionamento del programma senza scrivere codice. Supponiamo di aggiungere ad una scheda Cliente il numero e tipo di carta di credito.

In quasi tutti i programmi abbiamo la possibilità di aggiungere campi nelle tabelle del database.

In molti programmi un meccanismo di editing permette di modificare le finestre di input.

Il problema sembra quindi risolto, ma ci sono due punti non risolti:

- Abbiamo creato un **oggetto fuori standard** (vedi nelle pagine precedenti i problemi della personalizzazione ad oggetti)
- E soprattutto rimane aperta una domanda: **cosa ci facciamo con questa informazione ?**

Immaginiamo di volere praticare uno **sconto aggiuntivo** del 5% a chi ha una carta di credito VISA.

Per fare questo dobbiamo **modificare il codice sorgente** nel punto in cui vengono applicati gli sconti, con i problemi visti per la personalizzazione dei sorgenti.

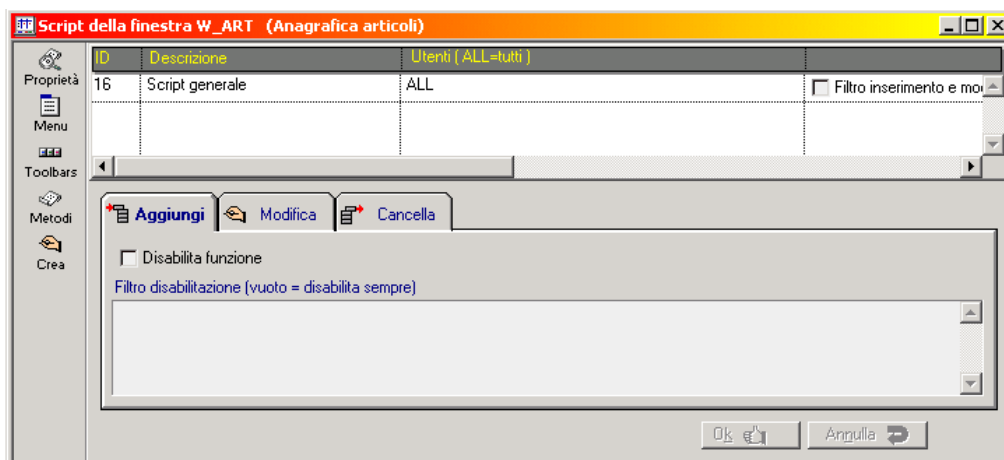
In Target cross, **basterà modificare la formula di calcolo** sconti tenendo conto di questo parametro, senza modificare niente altro. Oltre a ciò, il motore di personalizzazione consente **di modificare quanto visualizzato in una finestra senza creare un oggetto fuori standard**, come sarà illustrato nelle pagine successive.

I tre livelli della tecnologia di Target Cross

Target Cross offre tre distinte tecnologie di personalizzazione definite **Custom Tables**, **Instance editor** e **ROE (Reverse Object Engineering)**, in funzione dell'entità delle modifiche da apportare.

La filosofia di fondo è di utilizzare il sistema adatto rispetto alle esigenze, ottimizzando l'uso delle risorse.

La tecnologia Instance Editor

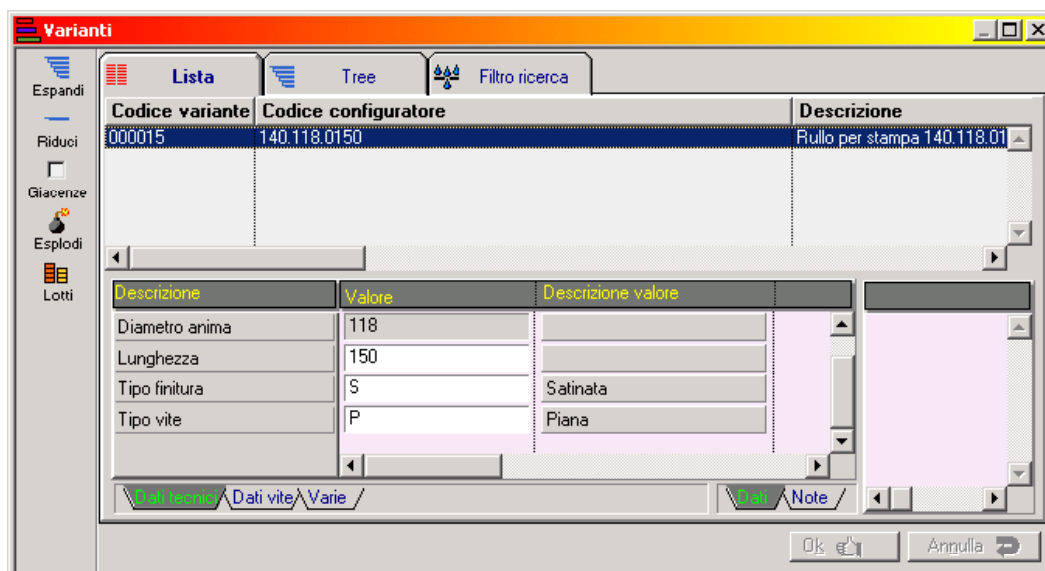


Si basa su una serie di **script**, creabili con drag and drop e scelte da lista, che consentono di modificare ogni finestra, menu o toolbar visibile a video.

Questi script possono essere **generici** o **specifici** per ogni utente o posto di lavoro.

Possono essere **aggiunti, tolti o disabilitati campi, menu o toolbar**. Nelle finestre a tab possono essere aggiunte pagine ai tab, su cui visualizzare i dati desiderati.

La tecnologia User Tables



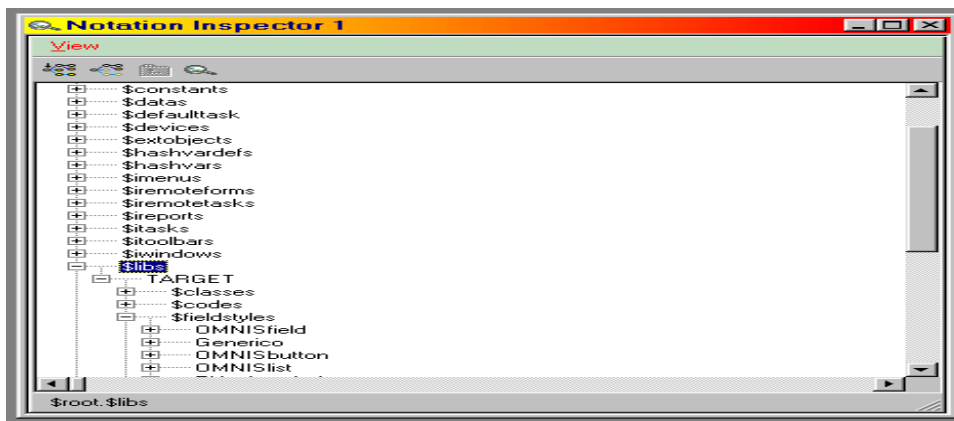
Le strutture dati possono essere modificate senza programmazione.

La visualizzazione e l'inserimento dei dati avvengono attraverso delle **strutture personalizzabili**, che possono dipendere anche dalla categoria merceologica, nel caso di articoli o di righe documenti.

Possiamo quindi avere dei **layout diversi nelle stesse finestre, a seconda di quello che vogliamo visualizzare**.

Questo ha reso possibile la creazione di un **Configuratore di prodotto** che, sfruttando le formula, raggiunge una flessibilità mai raggiunta prima nell'adattamento alle varie situazioni aziendali.

La tecnologia ROE (Reverse Object Engineering)



Dove i sistemi precedenti si rivelano insufficienti, la tecnologia ROE permette ogni tipo di personalizzazione intervenendo a livello del singolo oggetto o classe, e garantendo comunque l'automatica compatibilità con le versioni future.

Le modifiche vengono memorizzate in un repository presso l'Utente o lo sviluppatore e sono gestite dal **CRM (Change Repository Management)**.

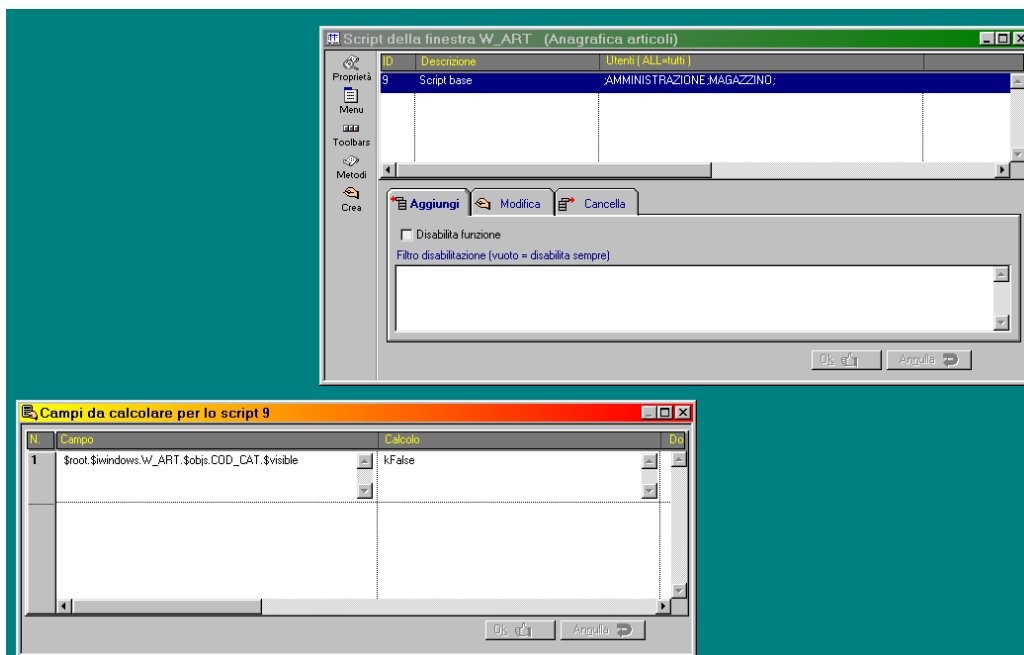
Il gestore **Heritage Manager** permette di creare copie ereditarie degli oggetti standard, che sommano le proprietà dello standard (e le successive modifiche) con le personalizzazioni apportate.

Analisi delle tecnologie

La tecnologia Instance Editor

Questa tecnologia ha lo scopo di modificare le instance delle finestre o di ogni altro oggetto del programma.

Quando una finestra (o un altro oggetto: menu, toolbar, ecc.) viene aperta, viene creata una **instance** della stessa, **presente solo in RAM**, che inizialmente sarà identica all'oggetto memorizzato nel programma, l'instance può essere modificata senza cambiare quella permanente.



La tecnologia Instance Editor permette di creare degli script, utilizzando il linguaggio di notation autoreferenziante, i quali possono cambiare ogni aspetto delle finestre e di ogni altro oggetto del programma.

In particolare questa tecnologia permette di **aggiungere tabs** alle finestre gestite in questo modo (quasi tutte quelle di una certa complessità), nei quali possono essere visualizzati gli oggetti desiderati.

Un modo molto funzionale di variare il contenuto delle finestre si ha **visualizzando nei tabs aggiunti delle subwindows**, che sono finestre create indipendentemente nel programma; questo consente di avere tutte le procedure di controllo a cui siamo abituati.

Gli script di finestra sono personalizzabili per ogni Utente, e quindi possiamo avere funzioni diverse a seconda dell'utilizzatore.

Oltre a ciò, possono essere eseguite le seguenti funzioni:

- **Nascondere o disabilitare** campi.
- **Filtrare i metodi di inserimento**, cancellazione o modifica delle tabelle. Possiamo fare in modo che l'Utente possa modificare o cancellare solo i records che soddisfano certi criteri.
- **Filtrare qualunque altro metodo** della finestra, che quindi può essere eseguito solo se vengono soddisfatti certi criteri.
- **Aggiungere o togliere menu** dalla barra in alto o dalle barre di finestra.
- **Personalizzare le toolbar** globali e quelle di finestra.
- **Personalizzare i metodi di controllo** della finestra o dei singoli oggetti
- **Togliere o disabilitare tabs** nelle finestre gestite in questo modo.
- **Modificare i metodi di controllo** dei campi e della finestra, che quindi possono modificare il loro comportamento.

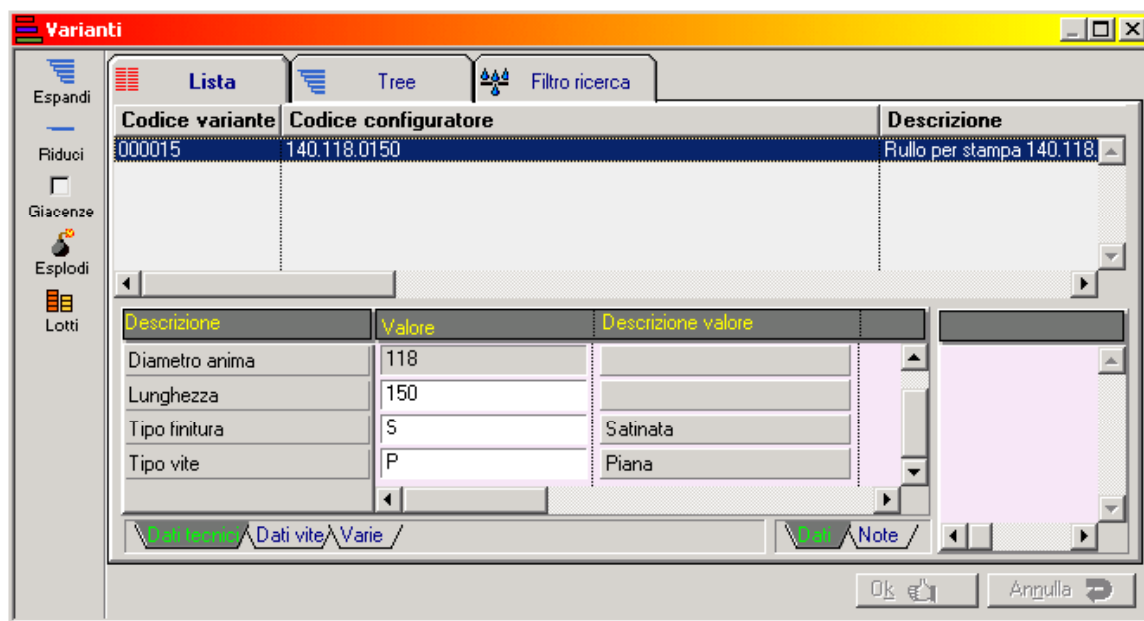
La tecnologia User Tables e il Configuratore di prodotto

Le tabelle di Target possono essere modificate senza operazioni di programmazione. Modificando le tabelle, il programma allinea automaticamente le tabelle presenti nel database con le modifiche.

La tecnologia User Table permette di inserire, modificare e visualizzare queste informazioni. La gestione a formule permette poi di utilizzare queste informazioni per determinare il comportamento del programma.

User Tables

La visualizzazione avviene attraverso strutture definibili dall'Utente, che possono interagire in modo sofisticato e possono mostrare dati molto diversi tra di loro.



I dati possono essere divisi in tre categorie: **dati di input**, **check box** e **campi note**, visualizzati in modo diverso. Possono poi essere divisi in sezioni, attivabili con tabs, in modo da non avere troppi dati sullo schermo.

Ogni campo di input può avere una **tabella di decodifica** del valore inserito, con la possibilità di selezione da lista.

I campi possono essere calcolati con una formula; possono avere una restrizione sui valori inseribili, anch'essa con formula; possono avere un valore di default, anche calcolabile a partire da altri campi.

Ogni campo può avere una formula di visualizzazione, ovvero essere visibile solo se questa è vera. Ad esempio se abbiamo il check box Tornitura, possiamo avere il campo Diametro tornitura visibile solo se Tornitura è selezionato, evitando input inutili.

Le strutture di input possono essere ereditarie a più livelli. Quindi se più strutture hanno una parte in comune, questa può essere creata una sola volta.

La cosa più interessante è che questi dati, una volta inseriti, possono effettivamente interagire con il resto del programma.

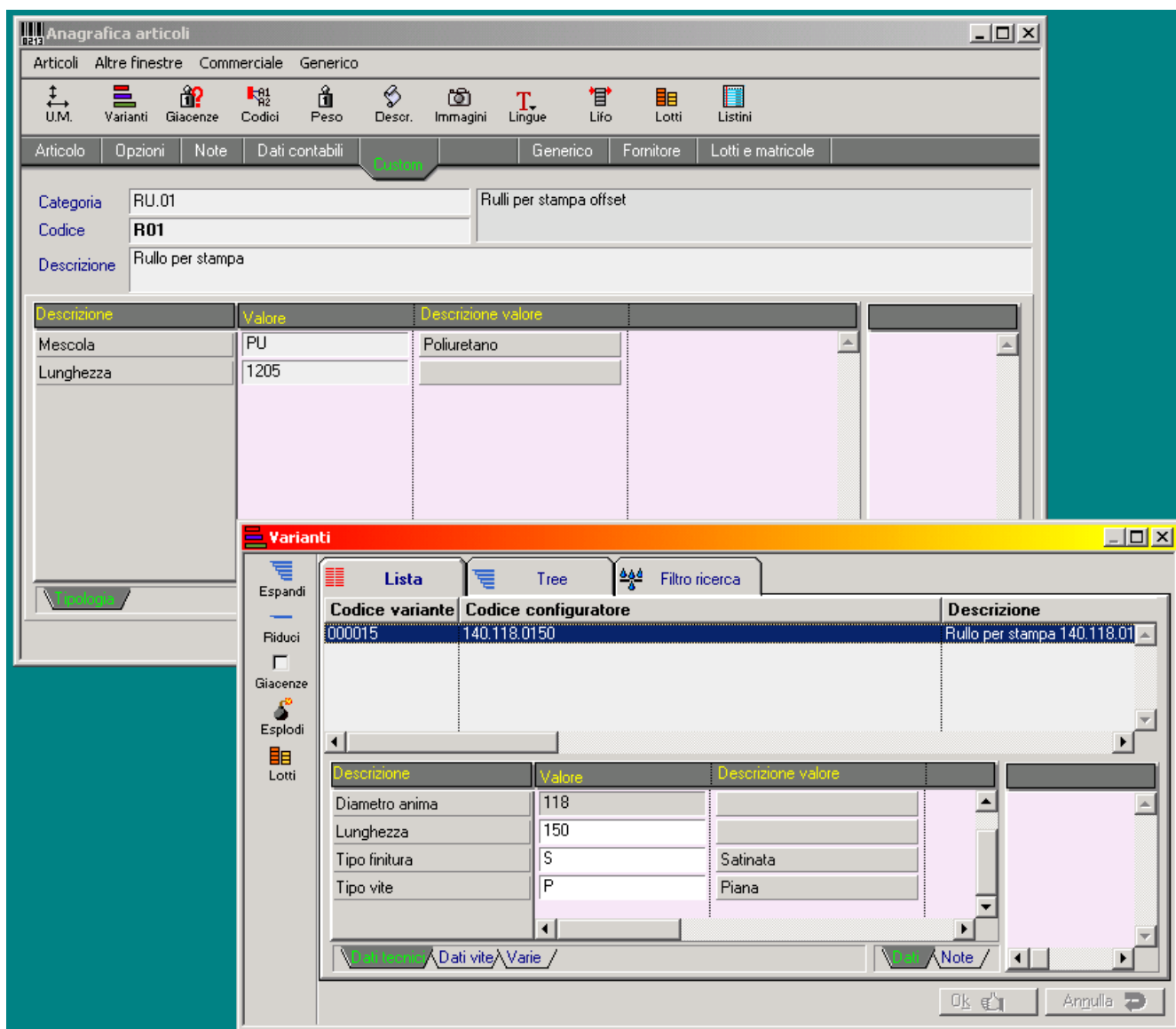
In Target Cross, questo compito viene assolto dalle formule ed in generale dagli elementi che costituiscono la tecnologia Spreadsheet. In questo caso, avendo la possibilità di inserire la formula dello sconto, questa sarà creata in modo tale da calcolare uno sconto del 5% nel caso desiderato.

Configuratore di prodotto

Uno delle applicazioni più brillanti della tecnologia delle User Tables, unita a quella Spreadsheet, si ha nel caso del **configuratore di prodotto** di Target Cross.

Queste due funzioni permettono di definire le strutture dati relative alle varianti delle più disparate categorie di prodotti, gestendone l'input e l'output, di calcolarne i prezzi e le distinte in base alla configurazione, di esplodere un prodotto configurato complesso nei suoi sottogruppi pure configurati (immaginiamo la porta e lo stipite, oppure l'armadio e lo sportello) calcolandone tutte le variabili.

L'archivio degli articoli è gestito a tre livelli (articolo generico, articolo e variante). La configurazione può avvenire ad uno qualunque di questi livelli. In particolare la possibilità di avere l'articolo suddiviso in varianti, ognuna con giacenze ed impegni, evita di dovere creare una nuova scheda articolo ogni volta che cambia qualche valore.



La **distinta dell'articolo** configurato può essere inserita a livello **dell'articolo** o **dell'articolo generico**, con ereditarietà. I vari componenti saranno configurati con le formule. In fase di lancio di produzione, **le formule di configurazione calcolano per tutti i livelli inferiori** le configurazioni necessarie, generando le varianti mancanti e lanciandole in produzione.

Anche i **cicli di lavorazione** possono venire calcolati in base alla configurazione.

E' possibile **attribuire dei valori a variabili che non generano una variante**, inserendoli nel documento creato (riga ordine o lancio di produzione). Possiamo quindi avere delle caratteristiche che non generano una variante, ma sono riferite alla singola commessa. Questi valori possono comunque venire utilizzati per il calcolo del prezzo, delle provvigioni, della distinta e dei cicli di lavoro.

La tecnologia ROE (Reverse Object Engineering)

Il sistema di personalizzazione a formule, pur nella sua forma più avanzata di array, funzioni custom e script, non può risolvere al 100 % i problemi di un'azienda. In particolare, una personalizzazione di solito comporta alcuni moduli funzionali:

- **Modifica della struttura del database**
- **Modifica delle strutture di input (finestre e calcoli collegati)**
- **Interazione con altre parti del programma**
- **Modifica della struttura di output (report e visualizzazione su finestre)**

Per gestire queste necessità è stata sviluppata la tecnologia ROE ((Reverse Object Engineering)

L'approccio ad oggetti, come è stato evidenziato precedentemente, si mostra inadeguato ad integrare nuove parti di software in un prodotto preesistente.

La tecnologia ROE è composta di due parti fondamentali:

- **Dynamic objects**
- **CRM (Change repository management)**

Il CRM a sua volta composto di:

- **Data method manager**
- **Schema manager**
- **Subwindow manager**
- **Heritage manager**

Per comprendere il funzionamento di questi meccanismi, è opportuno un cenno preliminare sulla struttura di Target Cross.

Analisi strutturale di Target Cross

La struttura del programma si presenta assai più articolata rispetto agli altri programmi Windows, anche ad oggetti.

Tipicamente in un programma a finestre l'unità di base è il cosiddetto "form", ovvero la finestra di input-output. Su di essa si trovano dei "controlli" che svolgono le funzioni necessarie.

Questo approccio "a form" è senz'altro comprensibile: cosa ha di diverso Windows? Ovviamente le finestre.

In pratica vediamo che in molti casi questa impostazione risulta insufficiente.

Target Cross ha una struttura costituita da un **numero maggiore di classi** rispetto agli altri programmi, ovvero:

- **Schemas**, le strutture delle tabelle
- **Tables**, i metodi associati a queste strutture
- **Queries**, ovvero joins predefiniti tra tabelle
- **Windows**, finestre classiche
- **Menus**, menu installabili sulla finestra dell'applicazione o sulle sottofinestre
- **Toolbars**, installabili sulla finestra dell'applicazione o sulle sottofinestre
- **Reports**, stampe
- **Searches**, filtri di ricerca
- **Codes**, metodi eseguibili non associati ad un oggetto visibile
- **Objects**, oggetti di tipo generico
- **Tasks**, unità funzionali in cui dividere l'applicazione

Già da questo elenco si possono intuire le molto maggiori potenzialità del prodotto. Per esempio gli oggetti Task permettono di dividere il programma in unità funzionali che condividono le stesse variabili, e di effettuare un **input dei dati multifinestra**.

Nei programmi concorrenti, l'input in un ambiente operativo (es. Schede articolo, schede Clienti, ordini) avviene quasi sempre su una singola finestra, non permettendo di sfruttare appieno le potenzialità grafiche. Confrontate ad esempio il nostro inserimento ordini con quello di qualunque concorrente per capire quello che intendiamo. Questo impedisce inoltre la sincronizzazione tra ambienti diversi.

Ognuna di queste classi (e non solo le finestre o forms) può avere dei metodi. Possiamo avere metodi associati alle strutture dati o agli oggetti di un report. Ad esempio un campo di un report può eseguire alcune azioni mentre viene stampato o se il valore cambia.

Questo tipo di impostazione ha permesso la creazione di un prodotto più solido, elegante e funzionale rispetto agli standard della concorrenza; oltre a ciò, viene **enormemente facilitata la creazione di modifiche**.

Ad esempio, il fatto che un report possa contenere le procedure per stamparsi o altro facilita l'aggiunta di stampe o export al programma. La struttura multifinestra, i cui più finestre condividono le variabili di un Task, permette l'aggiunta di finestre, menu o altri oggetti. Nei programmi standard, infatti, se aggiungiamo una finestra questa si trova ad essere "a tenuta stagna" rispetto alle altre, e non potrà interagire con esse. In Target Cross, ogni finestra può interagire con altre nello stesso Task, condividendone le variabili, lanciando e ricevendo messaggi, ecc.

La gestione per task è il presupposto indispensabile perché sia possibile effettivamente aggiungere funzionalità o elementi al programma. Immaginiamo ad esempio di volere aggiungere una finestra secondaria nell'ordine Cliente, che si riallaccia ai dati standard. Se questi dati sono contenuti in una finestra "a tenuta stagna", dovremo renderli disponibili "dall'interno", ma così facendo abbiamo modificato il programma standard, che è ciò che volevamo evitare.

La gestione a task invece consente un approccio semplificato alle nuove implementazioni; infatti, aggiungendo oggetti ad un task, questi automaticamente ne condividono le variabili.

Dynamic objects

Il processo di personalizzazione illustrato di seguito è reso possibile da una **fondamentale differenza** di Target Cross rispetto ai prodotti concorrenti.

Sviluppando un programma con un qualsiasi **linguaggio ad oggetti**, l'utilizzo degli stessi avviene ad esclusivo **beneficio del programmatore**; una volta che il programma è compilato, i vari oggetti sono fusi insieme in un codice oggetto non modificabile.

Possiamo vedere un programma classico come un insieme in cui i vari componenti sono "saldati" indissolubilmente tra di loro; in Target Cross invece essi sono "imbullonati", e possono essere in ogni momento smontati e sostituiti.

E' facile capire come questa differenza cambi decisamente l'approccio alle modifiche. Proviamo per un momento ad immaginare come sarebbe funzionale un'automobile in cui le varie parti, anziché smontabili, fossero tutte saldate insieme!

Il gestore di modifiche si incarica di tenere traccia dei cambiamenti apportati, salvandoli in un file separato (repository). In ogni nuova versione queste modifiche sono automaticamente riportate

Detto così sembra semplice, ma come tutte le cose di funzionamento semplice nasconde una tecnologia assai complicata e soprattutto è il frutto di un modo di pensare realmente innovativo.

Per potere dirimere gli innumerevoli conflitti che si presentano in una operazione di questo tipo, sono stati sviluppati diversi meccanismi. che nel loro insieme costituiscono il CRM (Change Repository Management).

CRM (Change Repository Management)

E' un gestore di modifiche, costituito da vari elementi.

- **Data method manager:** metodi di trattamento associati alle strutture dati.
- **Schema manager:** permette di modificare tutte le tabelle esistenti, anche quelle per le quali non è prevista una personalizzazione.
- **Subwindow manager:** permette di definire all'interno delle finestre delle aree personalizzate a disposizione dell' Utente.
- **Heritage manager:** crea copie ereditarie degli oggetti standard.

Inoltre il sistema di sviluppo utilizzato ci ha messo a disposizione alcuni strumenti particolarmente potenti:

- **Lists:** strutture dati a righe e colonne, visibili come variabili all'interno di tutto il programma. In pratica le liste di records non sono più legate al singolo form, ma sono utilizzabili ovunque.
- **Rows:** come sopra, ma con una sola riga, per trattare i dati record per record.
- **SQL aware objects:** le strutture dati di cui sopra possono essere **collegate ad una tabella o una query sul database**. Ogni modifica effettuata sulle variabili può essere **automaticamente riportata sul database**.
- **Smartlists:** le liste così definite si "accorgono" degli inserimenti, delle modifiche e delle cancellazioni effettuate, e possono riportarle in automatico sul database.

Data method manager

Questa funzione permette di inserire **metodi associati alle strutture dati**. Per esempio nella struttura dati (tabella) una riga ordine (e non nella finestra di inserimento) possiamo inserire un metodo \$insert che si occupa di controllare la validità dai dati, inserire la riga e aggiornare l'ordinato dell'articolo. In questo modo il metodo sarà a disposizione in qualunque parte del programma in cui vogliamo inserire una riga di ordine. Per attivare questo metodo, supponiamo di avere una tabella RIGA_ORDINE; definiremo una list o una row (a seconda se vogliamo inserire una riga alla volta o molte insieme) al livello che vogliamo (locale, di window, di task o globale) e la linkeremo alla tabella con il comando

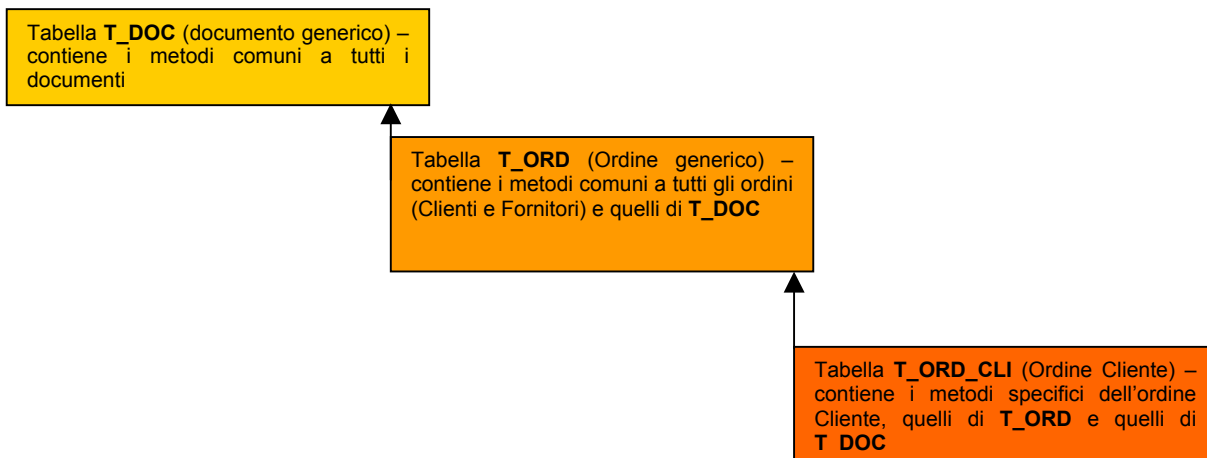
```
Do MyList.$definefromsqlclass('RIGA_ORDINE')
```

In questo modo nella lista avremo tutte le colonne della riga ordine, e se per esempio abbiamo inserito nella tabella un metodo \$insert che effettua l'inserimento, sarà sufficiente il comando

```
Do MyList.$insert()
```

Per effettuare l'inserimento in ogni parte del programma.

I metodi associati alle strutture dati sono ereditari. Tabelle specifiche ereditano i metodi da tabelle più generiche. Ad esempio nel caso degli ordini Clienti abbiamo



A qualunque livello di questa catena **l'Heritage Manager** (vedi oltre) può inserire un ulteriore livello di ereditarietà, con una Table personalizzata. In questo modo possiamo cambiare il comportamento di tutto quello che sta a valle della nostra modifica: più questa è a monte, più ampio sarà l'effetto delle variazioni.

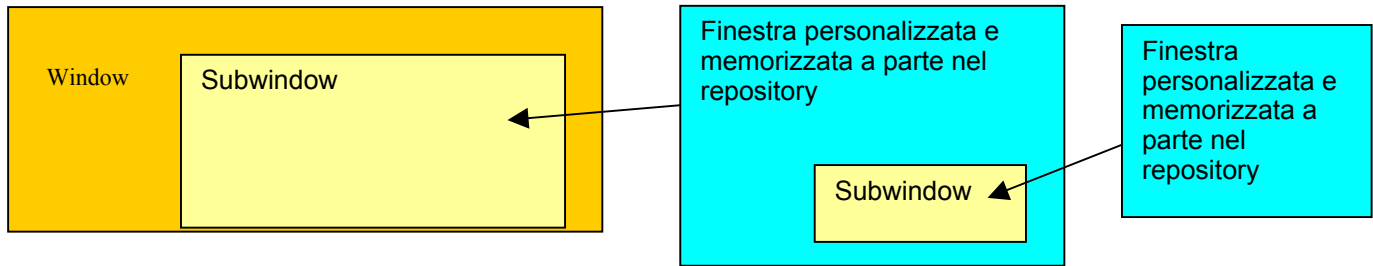
Schema manager

Questa funzione permette di apportare **le modifiche desiderate a tutte le tabelle del programma**, e di riportarle nelle versioni successive. Questo significa che se nel frattempo nel programma standard sono stati aggiunti dei campi in una tabella, avremo questi ultimi e quelli aggiunti in modo custom.

L'aggiornamento avviene rispettando la sequenza dei campi memorizzati nel database, per cui non è mai necessario fare un import – export dei dati, ma è sufficiente aggiornare il database con un ALTER TABLE (gestito in modo automatico):

Subwindow manager

Le subwindows sono aree di finestra dove è possibile vedere il contenuto di un'altra finestra, che a sua volta può contenerne un'altra e così via.



Questa funzione è molto utile perché le finestre visibili come subwindow sono oggetti esterni al programma, ma ne vengono incorporati condividendone le variabili.

Per visualizzare una tabella aggiuntiva, sono senz'altro il sistema più immediato e funzionale. Nelle tabelle USER predefinite nel programma, sono già previste nelle finestre delle subwindow, denominate Window_USER, dove visualizzare queste estensioni dei dati.

Per non sottrarre spazio alle finestre, le subwindow sono contenute in Tabs visibili solo se attivate. Il sistema di **messaging multilivello** consente alle subwindow di ricevere e di inviare messaggi.

Heritage manager

Con questa funzione possiamo **creare copie ereditarie degli oggetti standard**, con possibilità di personalizzarli in tutto o in parte. A differenza della sostituzione di moduli nei programmi tradizionali, questo sistema consente, ove desiderato, di continuare ad usufruire della procedure standard.

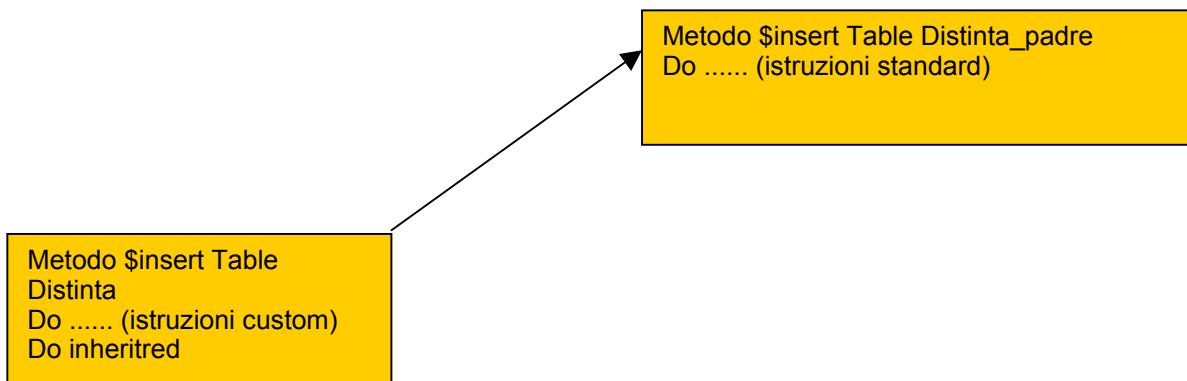
Avendo a disposizione molte più classi rispetto agli altri programmi, questa possibilità si applica non solo alle finestre, ma a tutte le classi del programma, ed è particolarmente utile per quanto riguarda i metodi ed in combinazione con il Data method manager.

Supponiamo ad esempio di volere creare una procedura di inserimento di un componente di distinta che faccia dei calcoli particolari prima dell'inserimento. Verrà creata una table **DISTINTA**, mentre la table originale sarà rinominata **DISTINTA_FATHER**.

La table **DISTINTA** erediterà i metodi; per sostituirli, ne creeremo uno con lo stesso nome.

Se ad esempio la tabella padre ha un metodo \$insert(), potremo bypassarlo inserendone nella classe figlia uno con lo stesso nome.

Se in realtà vogliamo solo aggiungere delle nostre righe di codice al metodo standard, possiamo eseguire il metodo della table padre con il comando Do inherited. Il nostro metodo potrebbe quindi essere



Conclusioni

L'insieme dei meccanismi di personalizzazione analizzate nelle pagine precedenti costituisce un sistema di sviluppo con una potenzialità senza precedenti per adattare il programma alle necessità degli Utenti.

Il principio ispiratore è quello di operare con le modifiche al più basso livello necessario; ecco perché sono presenti vari meccanismi di modifica, ognuno dei quali permette di risolvere problemi di entità diversa, con il minimo dispendio di energie ed i minimi costi di mantenimento.

La gestione a formule di Target Cross rimane tuttavia la vera caratteristica qualificante del prodotto, in quanto essa consente da un lato di evitare un gran numero di personalizzazioni, e dall'altro è l'anello di congiunzione tra le modifiche ed il codice sorgente, permettendo di modificare il comportamento del programma senza modificare quest'ultimo.