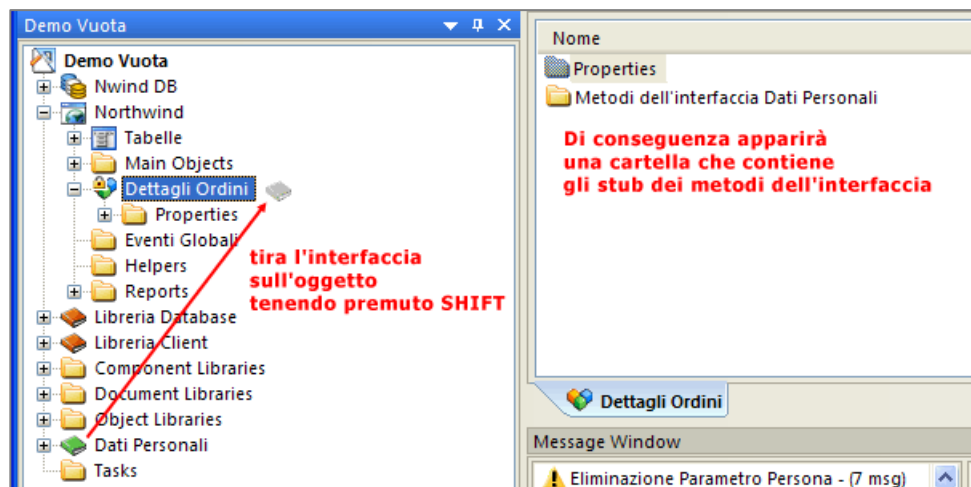




Come implementare un'interfaccia

Per eliminare l'implementazione di una interfaccia da parte di una classe o documento, occorre tirarla su di esso tenendo premuto il tasto *ctrl*. Questa operazione causerà la rimozione dei metodi corrispondenti all'interno della classe o documento.

Se si modifica la definizione dell'interfaccia o dei metodi che essa contiene, tutte le modifiche si rifletteranno automaticamente sugli oggetti corrispondenti nelle classi o documenti che la implementano. Infine, implementando un'interfaccia in una classe base, anche le derivate la implementeranno e sarà possibile specializzarne i metodi.

### 5.11 Sincronizzazione di documenti

L'architettura delle applicazioni web rende particolarmente semplice riuscire a mettere a disposizione servizi applicativi condivisi fra un grande numero di persone. La semplicità deriva dal fatto che esse prevedono un unico application server (anche in modalità cluster) e un unico database server. Quindi i dati sono sempre consolidati e aggiornati in tempo reale.

Con la diffusione delle architetture OWA (Offline Web Application) tutto questo cambierà. Una OWA è quasi l'opposto logico di una web application classica, infatti essa deve funzionare completamente nel dispositivo client, senza collegamento in rete. Questo comporta che sia i dati che la logica applicativa devono funzionare in locale, disconnessi dal server centrale, e che il problema della sincronizzazione diventa importante e centrale per il successo dell'intero sistema informativo.

Gestire la sincronizzazione di più database distribuiti non è semplice; non è solo un problema di replica, dove si tengono allineati più database server allo scopo di aumentare l'affidabilità del sistema. Vediamo le principali problematiche:

- 1) *Parzializzazione*: solo una piccola parte dei dati deve essere copiata in ogni terminale locale, quelli che competono all'utente che attualmente lo sta usando.
- 2) *Concorrenza*: i dati possono essere modificati contemporaneamente da più terminali, quindi bisogna gestire i conflitti in fase di sincronizzazione.
- 3) *Sicurezza*: i terminali possono essere facilmente compromessi, quindi il dato in arrivo deve essere rivalidato secondo le regole del processo anche durante la fase di sincronizzazione.
- 4) *Velocità*: la larghezza di banda dei terminali può essere anche molto ridotta, occorre implementare un sistema di sincronizzazione differenziale, che permetta di minimizzare il traffico dati.
- 5) *Real Time*: la sincronizzazione deve avvenire anche in tempo reale, se è disponibile la connessione, per permettere un feedback immediato delle operazioni.

Per poter affrontare in maniera efficace e sicura queste problematiche, dalla versione 10.5 Instant Developer contiene un nuovo servizio ai documenti denominato *sincronizzazione* che aziona un framework dedicato all'allineamento di dati fra database distribuiti. Nei paragrafi successivi vengono illustrati i principi di funzionamento della sincronizzazione, ma fin d'ora è importante notare che non si tratta di allineamento fra database a livello di singoli record, ma di una comunicazione a livello di classi di documenti. Senza l'utilizzo della document orientation non sarà quindi possibile l'utilizzo del framework di sincronizzazione di Instant Developer.

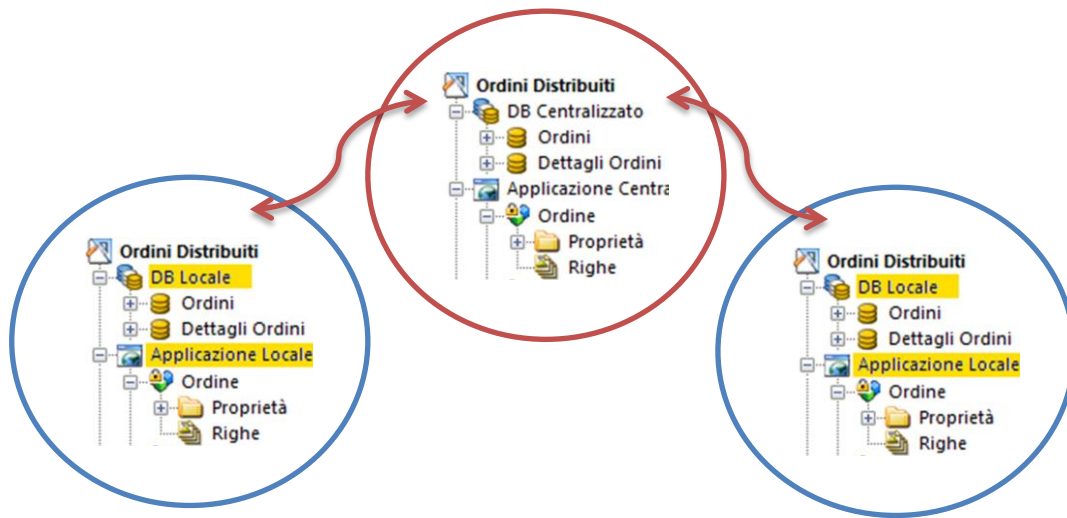
#### 5.11.1 L'architettura di riferimento

Per poter esemplificare il funzionamento della sincronizzazione, immaginiamo di sviluppare un'applicazione che permette di aggiornare l'elenco degli ordini in maniera distribuita. Avremo quindi un server centrale, in cui l'elenco degli ordini risiede in un database centrale, e due terminali secondari, anch'essi aventi un database locale con una parte della lista degli ordini.

Siccome tutto avviene tramite la document orientation, in ogni applicazione ci saranno anche i documenti *Ordine* e *Riga Ordine* per il trattamento dei dati relativi. Lo schema alla pagina seguente visualizza la situazione descritta.

Dobbiamo quindi gestire questa situazione: tre database e tre applicazioni distinte. Supponiamo che ad un determinato istante i database siano allineati e ognuno contenga la parte dei dati giusta. Quello centrale tutti gli ordini, i database locali quelli che l'utente collegato deve gestire.

**Attenzione:** il framework è in grado di sincronizzare documenti omologhi, cioè con lo stesso nome di classe. Le proprietà che hanno lo stesso nome vengono sincronizzate automaticamente, le altre vengono riconciliate tramite l'uso di *named properties*.



Per attivare il framework di sincronizzazione sono richieste tre condizioni:

- 1) Aver attivato il servizio di sincronizzazione nella videata delle proprietà delle classi documento che devono essere allineate.
- 2) Almeno un documento da sincronizzare per ogni applicazione deve essere collegato ad una tabella di un database.
- 3) Non utilizzare una versione Express di Instant Developer, per la quale il framework di sincronizzazione non è disponibile.

### 5.11.2 Raccolta delle differenze lato client

Vediamo adesso cosa succede nelle applicazioni quando si trovano nelle condizioni predette. Innanzitutto possiamo notare che il framework crea una nuova tabella di appoggio, chiamata ZZ\_SYNC, nei database del progetto. Tale tabella è destinata a contenere le variazioni avvenute ai documenti dell'applicazione.

| Ordini Distribuiti |                                                           |
|--------------------|-----------------------------------------------------------|
| DB Locale          |                                                           |
| ZZ SYNC            |                                                           |
| ID                 | Numero della variazione                                   |
| Doc Dna            | Identificativo del documento variato, completo dei parent |
| Domain             | Document Domain                                           |
| Stato              | Tipo di modifica                                          |
| Valori             | Contenuto della modifica                                  |
| Data Ora           | Istante in cui è avvenuta                                 |

Ogni volta che l'applicazione salva un documento, sia perché l'utente lo ha variato dall'interfaccia utente che a causa del codice eseguito, le modifiche vengono memorizzate nella tabella ZZ\_SYNC, un record per ogni documento o sotto-documento modificato. Questo vale sia per le operazioni di inserimento, che di modifica e cancellazione e avviene sia nei database locali che in quello centrale.

**Attenzione:** se si effettuano modifiche al database direttamente tramite query, queste non verranno memorizzate nella tabella ZZ\_SYNC e quindi non potranno essere sincronizzate.

Nella tabella possiamo notare il campo *Domain*: esso contiene una stringa che identifica il dominio del documento, quello cioè a cui appartiene, chi può gestirlo. La scelta del dominio è un problema applicativo; nell'esempio della lista ordini il dominio può essere rappresentato dal campo *ID Impiegato* della tabella *Ordini*, che rappresenta l'utente che ha creato l'ordine e quindi può gestirlo.

Il framework di sincronizzazione controlla il documento in fase di salvataggio per verificare se esso supporta una proprietà che esprime il concetto DO\_DOMAIN; se non la trova notifica al documento l'evento OnGetNamedPropertyValue passando come parametro DO\_DOMAIN. Se l'evento non risponde, si suppone che il documento non sia associato ad un dominio e quindi deve essere sincronizzato verso ogni terminale che ne fa richiesta. Vediamo un esempio di codice relativo al documento ordine.

```
event Ordini.OnGetNamedPropertyValue(  
    string PropertyName  
    inout string PropertyValue  
)  
{  
    if (PropertyName == "DO_DOMAIN")  
    {  
        PropertyValue = toString(Idimpiegato)  
    }  
}
```

### 5.11.3 Ciclo di sincronizzazione

Vediamo ora come si configura e si innesca il ciclo di sincronizzazione, operazione che avviene sempre a partire dal terminale remoto verso il server centrale. Vediamo il codice che deve essere scritto:

```
public void GestioneOrdini.ButtonSync()  
{  
    SyncService.serverURL = "http://www.myserver.com/remoteapp/remoteapp.aspx"  
    SyncService.username = "username"  
    SyncService.password = "password"  
    SyncService.synchronize()  
}
```

Le prime tre righe rappresentano le impostazioni iniziali del servizio di sincronizzazione e possono essere eseguite una volta per tutte negli eventi di attivazione della sessione. La quarta riga è l'unica necessaria ad avviare la sincronizzazione vera e propria. Siccome l'operazione può richiedere un certo tempo, essa può essere eseguita anche in una server session per evitare di bloccare l'interfaccia utente. Se invece la si esegue nella sessione browser, allora gli eventuali documenti già presenti nell'interfaccia utente verranno automaticamente aggiornati se la sincronizzazione li ha modificati.

Il ciclo di sincronizzazione si svolge nelle seguenti fasi:

- 1) Il terminale client, che inizia la sincronizzazione, raccoglie tutte le variazioni memorizzate nella tabella ZZ\_SYNC e le invia al server.
- 2) Il server elabora i dati ricevuti, aggiorna i documenti server, poi prepara l'elenco delle variazioni che il client deve ricevere in funzione di tutte le altre sincronizzazioni e delle modifiche avvenute nel server centrale stesso.
- 3) Il terminale client riceve la lista delle variazioni che deve apportare al database locale, le esegue ed eventualmente aggiorna l'interfaccia utente. Al termine cancella il contenuto della tabella ZZ\_SYNC perché già inviato al server stesso.

Vediamo ora cosa avviene nel server quando riceve i dati di sincronizzazione.

- 1) Per prima cosa viene notificato all'applicazione server l'evento OnSynchronize che permette di calcolare, in base allo user name e alla password, il dominio di riferimento per il terminale che ha richiesto la sincronizzazione. In questo modo verranno sincronizzati solo i documenti che appartengono a tale dominio, oppure quelli che non supportano nessun dominio. L'evento permette anche di interrompere l'intera operazione di sincronizzazione se l'utente non viene riconosciuto.
- 2) Se l'operazione può proseguire, allora vengono caricati dal database centrale tutti i documenti per i quali il terminale client ha inviato una variazione, rispettando la stessa struttura di modifica avvenuta nel client. Se, ad esempio, l'utente ha modificato una riga dell'ordine, nel server verrà caricato l'intero ordine e non solo la riga.
- 3) Per ogni documento caricato, si verifica che il dominio ricalcolato sul server sia corrispondente a quello calcolato dall'evento OnSynchronize, per evitare di aggiornare documenti fuori dominio.
- 4) A questo punto le variazioni comunicate dal client vengono applicate al documento.
- 5) Quando tutte le variazioni applicate dal client sono state applicate all'insieme dei documenti caricati in memoria, è possibile iniziare la fase di validazione e salvataggio degli stessi. L'evento OnValidate viene notificato a tutti i documenti coinvolti specificando 10 come parametro Reason. In questo modo il documento sa che sta per essere salvato a causa di una sincronizzazione e può eseguire i controlli più opportuni.

- 6) Tutti i documenti che hanno passato la validazione vengono salvati sul database, in più fasi per superare problemi di dipendenze reciproche. Al termine del ciclo di salvataggio uno o più documenti potrebbero non aver passato la validazione a causa di errori, altri potrebbero non essere stati salvati sul database ed infine altri ancora potrebbero venire ulteriormente modificati durante il ciclo di salvataggio stesso.
- 7) Il server inizia quindi la raccolta di tutte le variazioni che deve inviare al client. Esse possono comprendere sia variazioni avvenute nel server in funzione della sincronizzazione di altri client e delle stesse operazioni server, sia tutte le casistiche elencate al punto precedente. Queste variazioni e la lista degli errori vengono inviati come risposta al client che li elabora come di seguito dettagliato.

La fase di sincronizzazione lato client a seguito della risposta del server è sostanzialmente identica a quella che avviene lato server, viene solamente usato il valore 11 come parametro Reason dell'evento OnValidate. Al termine dell'applicazione delle variazioni ricevute dal server, l'interfaccia utente viene aggiornata, mostrando così anche eventuali errori comunicati dal server, al termine viene cancellato il contenuto della tabella ZZ\_SYNC nel client.

**Attenzione:** il framework di sincronizzazione non cancella mai il contenuto della tabella ZZ\_SYNC lato server in quanto esso è necessario per sincronizzare ulteriori client che si collegano ad intervalli temporali non conosciuti. Si consiglia tuttavia di predisporre una funzione applicativa che permetta di cancellare i dati più vecchi, ad esempio, di un mese o comunque dell'intervallo di tempo massimo dopo il quale i client non possono più sincronizzarsi in maniera differenziale.

#### 5.11.4 Re-sincronizzazione e query remote

Nei paragrafi precedenti abbiamo supposto che i database fossero inizialmente in uno stato allineato. Ma quando un client si collega per la prima volta, oppure se si sincronizza solo dopo un intervallo di tempo molto grande, come è possibile ottenere una sincronizzazione completa dei documenti? Per risolvere questo problema sono state previste tre soluzioni.

Il primo meccanismo di re-sincronizzazione avviene quando il server si accorge che il client non ha mai eseguito la sincronizzazione, oppure che ha perso dei record di sincronizzazione differenziale perché non si è collegato da tanto tempo. In questo caso viene notificato l'evento OnResyncClient ad ogni documento per il quale è stato attivato il servizio di sincronizzazione. Il documento può impostare dei criteri di filtro che verranno usati per caricare la collection dei documenti da inviare al client, oppure direttamente caricare lui la collection desiderata. Vediamo un esempio di queste operazioni nell'applicazione di gestione ordini vista prima.

```
event Ordini.OnResyncClient(  
    IDCollection Collection of IDDocument  
    string ClientDomain  
    date time LastSync1  
    inout boolean Skip  
)  
{  
    Idimpiegato = convert(ClientDomain)  
}
```

Per il documento ordine è molto semplice. Siccome il dominio del documento viene rappresentato dall'ID dell'impiegato che lo deve gestire, l'unico filtro per inviare la lista degli ordini al terminale client è quello sull'ID dell'impiegato. Da notare che il parametro ClientDomain è proprio il dominio attribuito dal server in funzione dello UserName e Password comunicati dal client.

Per il documento riga ordine è più complicato perché la re-sincronizzazione considera i documenti in modo piatto e non strutturato. Un esempio di codice è il seguente:

```
event DettagliOrdini.OnResyncClient(  
    IDCollection Collection of IDDocument  
    string ClientDomain  
    date time LastSync1  
    inout boolean Skip  
)  
{  
    if (ClientDomain != "")  
    {  
        Skip = true  
        //  
        int imp = toInteger(ClientDomain)  
        //  
        Recordset rs = new()  
        //  
        select into recordset (rs)  
        set Idordine = Idordine  
        set Idprodotto = Idprodotto  
        set PrezzoUnitario = PrezzoUnitario  
        set Quantita = Quantita  
        set Sconto = Sconto  
        from  
        DettagliOrdini // master table  
        where  
        Idordine in subquery  
        select //  
        Idordine  
        from  
        Ordini // master table  
        where  
        Ordini.Idimpiegato = imp  
        this.loadCollectionFromRecordset(Collection, rs, 0)  
    }  
}
```

In questo caso il caricamento della collection di tutte le righe ordine da inviare al terminale viene fatto nell'evento stesso, selezionando quelle appartenenti a tutti gli ordini che l'impiegato può gestire. Da notare anche che il parametro *Skip* è stato impostato al valore *true* in modo da saltare il caricamento automatico da parte del framework.

In altre situazioni può essere il terminale client a richiedere una sincronizzazione di una o più classi di documento, magari perché non si è sicuri del corretto allineamento delle stesse. Per richiedere la re-sincronizzazione è sufficiente usare la procedura ResyncClass prima del metodo Synchronize. Vediamo un esempio di codice che richiede l'intera lista degli ordini gestibili da un certo terminale.

```
 public void GestioneOrdini.ButtonResync()  
{  
     SyncService.resyncClass(Ordini.className())  
     SyncService.resyncClass(DettagliOrdini.className())  
     SyncService.synchronize()  
}
```

L'ultimo metodo di re-sincronizzazione è adatto ad un caricamento massivo di dati dal server al client. Immaginiamo ad esempio di dover allineare per la prima volta una tabella anagrafica con un milione di record. L'uso dei metodi visti prima potrebbe essere molto oneroso, quindi è stata prevista la possibilità di eseguire una query remota, tramite il metodo Query della libreria SyncService.

Questa funzione non è pensata per eseguire una qualunque query SQL nel server perché sarebbe piuttosto insicuro, tuttavia permette di inviare un comando parametrico che il server analizza ed esegue se di suo gradimento. Il risultato di una query remota è un Recordset che può essere manipolato direttamente dal codice del client.

Quando il server riceve una richiesta di query, esso notifica all'applicazione l'evento OnSyncQuery, sempre dopo aver autenticato il client tramite l'evento OnSynchronize. Nella pagina seguente viene mostrato un esempio di codice che restituisce una lista di ordini in funzione del cliente richiesto.

L'utilizzo delle query remote per il caricamento massimo è semplice; ad esempio se il client si accorge che la tabella da caricare è vuota, può iniziare ad eseguire query remote chiedendo ogni volta una parte dei dati per non sovraccaricare il sistema.

Un'altra occasione di utilizzo delle query remote è quando il cliente deve recuperare un'informazione aggiornata in tempo reale ed è disponibile una connessione, anche senza passare dal sistema di sincronizzazione dei documenti.

```
event RemoteClient.OnSyncQuery(  
    string Command  
    IDArray Params  
    string Domain  
    Recordset RS  
)  
{  
    if (Command == "ordini")  
    {  
        Recordset r = new()  
        string cli = ""  
        //  
        if (Params.length() > 0)  
            cli = Params.getValue(0)  
        //  
        select into recordset (r)  
        Ordini.Idordine as IDOrdine  
        Ordini.Idcliente as IDCliente  
        Clienti.NomeSocieta as RagioneSociale  
        from  
        Ordini  
        Clienti  
        where  
        Ordini.Idimpiegato = toInteger(Domain)  
        Ordini.Idcliente like cli + "%"  
        //  
        RS.copyFrom(r)  
    }  
}
```

#### 5.11.5 Gestione dei conflitti

Abbiamo visto che in base alla parzializzazione dei documenti in diversi domini è possibile evitare che più persone modifichino contemporaneamente lo stesso documento, in quanto solo una di esse lo ha a disposizione nel proprio database locale. Tuttavia una soluzione preventiva di tutti i possibili conflitti non è sempre possibile, quindi è necessario poterli gestire quando essi si presentano.

Il framework di sincronizzazione di Instant Developer consente di farlo in due modi complementari, che devono essere attivati solo se effettivamente necessario in quanto aumentano il consumo di risorse destinato alla loro gestione.

Il primo modo di gestire i conflitti è quello di attivare un sistema di lock ottimistico per un certo tipo di documenti, chiamando la procedura EnableOptimisticLock sia nei client che nel server. Quando questo avviene, un client può sincronizzare le variazioni al documento solo se nel frattempo non ne sono avvenute delle altre in precedenza, nel qual caso si ottiene un errore. La granularità delle modifiche è a livello di singola proprietà del documento, quindi due utenti possono modificare contemporaneamente parti

diverse dello stesso documento senza che questo causi problemi. L'attivazione del lock ottimistico raddoppia il consumo di risorse di sincronizzazione perché è necessario memorizzare sempre il valore originale delle proprietà in fase di modifica e non solo quello attuale.

Sono inoltre disponibili i metodi GetLock e ReleaseLock della libreria SyncService, che consentono di inoltrare una richiesta di lock per un documento al server, permettendo in tal modo di creare un meccanismo di lock preventivo, o pessimistico, che in alcune situazioni applicative è preferibile a quello ottimistico. Questi metodi però richiedono una connessione internet attiva, senza la quale la richiesta di lock non può essere inoltrata.

#### **5.11.6 Ulteriori metodi di SyncService**

In questo paragrafo vengono illustrati ulteriori metodi del framework di sincronizzazione utili in alcune situazioni particolari.

- 1) ResyncDocument: permette di ottenere al volo la versione aggiornata di un documento. Il documento viene aggiornato senza essere salvato.
- 2) ResyncCollection: permette di ottenere al volo la versione aggiornata di tutti i documenti di una collection. La collection viene aggiornata senza essere salvata.
- 3) LastSynchronization: restituisce la data e l'ora dell'ultima sincronizzazione effettuata dal client.
- 4) DontSync: questo metodo di *IDDocument* permette di specificare quali proprietà di un documento non devono essere sincronizzate. Solitamente esso viene utilizzato nell'evento OnGetNamedPropertyValue del documento quando viene chiamato con parametro `DO_DONTSYNC`. Escludere una proprietà di un documento dalla sincronizzazione permette di risparmiare risorse lato client e di proteggere dati privati lato server.

#### **5.12 Domande e risposte**

La Document Orientation rende semplice creare applicazioni enterprise anche di tipo SOA, ma il framework che ne permette il funzionamento è piuttosto complesso ed è stato possibile descrivere solo le interazioni di primo livello.

Se in qualche caso non è chiaro come affrontare una specifica problematica ti invito ad inviare una domanda via mail [cliccando qui](#). Prometto una risposta a tutte le mail nel tempo a mia disposizione; le domande più interessanti e frequenti verranno pubblicate all'interno di questo paragrafo nelle successive edizioni di questo libro.