



**Multiutenza+Persistenza+Sincronizzazione
=Conflitti**

**Conflitti, solo quando sincronizzi due DB?
e quando persisti oggetti ottenuti da una API REST?**



SPEAKER: MAURIZIO DEL MAGNO



MAURIZIO DEL MAGNO DEVELOPER



Lev@nte software

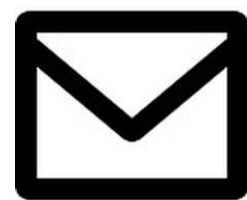


i-ORM

github.com/mauriziodm/iORM

DJSON

github.com/mauriziodm/DJSON



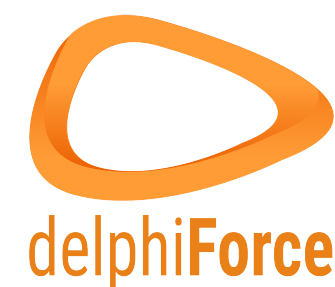
mauriziodm@levantesw.it

mauriziodelmagno@gmail.com



facebook.com/maurizio.delmagno

iORM + DJSON (group)



Membro fondatore

eInvoice4D

<https://github.com/delphiforce/eInvoice4D>



**Multiutenza+Persistenza+Sincronizzazione
=Conflitti**

Conflitti, solo quando sincronizzi due DB?
e quando persisti oggetti ottenuti da una API REST?



SPEAKER: MAURIZIO DEL MAGNO

Perchè **una** Replica?

Perché voler (o dover) scegliere di avere un DB locale?

Perchè **una** Replica?

Perché voler (o dover) scegliere di avere un DB locale?

Disponibilità

- **Mantenimento UX** con Connessione non garantita
- **Fault tolerance**
- **Superamento single point of failure**

Perchè **una** Replica?

Perché voler (o dover) scegliere di avere un DB locale?

Disponibilità

Perchè **una** Replica?

Perché voler (o dover) scegliere di avere un DB locale?

Disponibilità

Prestazioni / Scalabilità

- Località del dato (Edge computing)
- Sedi distaccate (latenza)
- Elaborazione mobile (latenza)
- Distribuzione e bilanciamento
- Caching (REDIS)
- Database ibridi (RDBMS + NoSQL)

Perchè **una** Replica?

Perché voler (o dover) scegliere di avere un DB locale?

Disponibilità

Prestazioni / Scalabilità

Perchè **una** Replica?

Perché voler (o dover) scegliere di avere un DB locale?

Microservizi

- Ogni microservizio ha il suo Stack e il suo DB (frammentazione?)
- Geograficamente distribuiti
- Team distinti e indipendenti
- Strumenti differenti (Tecnologie, linguaggi, framework)

Disponibilità

Prestazioni / Scalabilità

Perchè **una** Replica?

Perché voler (o dover) scegliere di avere un DB locale?

Disponibilità

Prestazioni / Scalabilità

Microservizi

Perchè **una** Replica?

Perché voler (o dover) scegliere di avere un DB locale?

Sicurezza

- App. Remota senza connessione
- Sincronizzazione solo in sede (LAN)
- Nessuna connessione con l'esterno

Disponibilità

Prestazioni / Scalabilità

Microservizi

Perchè **una** Replica?

Perché voler (o dover) scegliere di avere un DB locale?

Disponibilità

Prestazioni / Scalabilità

Microservizi

Sicurezza

Perchè **una** Replica?

Perché voler (o dover) scegliere di avere un DB locale?

Pervasività

- Pervasive computing

Integrazione nell'ambiente circostante di dispositivi/sensori che, grazie alle loro capacità computazionali e di comunicazione, fanno in modo che le persone li possano agevolmente sfruttare/utilizzare con una maggiore interazione con l'ambiente stesso

(Mark Weiser, Xerox Palo Alto Research Center)

Le tecnologie più profonde sono quelle che scompaiono, si intrecciano nel tessuto della vita quotidiana fino a diventare indistinguibili da essa

(Mark Weiser, Xerox Palo Alto Research Center)

- Connettività non garantita

- Memorizzazione locale e invio quando possibile (sincronizzazione)

Disponibilità

Prestazioni / Scalabilità

Microservizi

Sicurezza

Perchè **una** Replica?

Perché voler (o dover) scegliere di avere un DB locale?

Disponibilità

Prestazioni / Scalabilità

Microservizi

Sicurezza

Pervasività

DatabaseDistribuito

DatabaseDistribuito

Definizione

Insieme di database fisicamente distribuiti su diversi dispositivi (nodi) interconnessi tra loro (rete), i cui dati sono inter-relazionati dal punto di vista logico

Dal punto di vista dell'operatore **appare come un unico database** (trasparenza)

DatabaseDistribuito

Definizione

Insieme di database fisicamente distribuiti su diversi dispositivi (nodi) interconnessi tra loro (rete), i cui dati sono inter-relazionati dal punto di vista logico

Dal punto di vista dell'operatore **appare come un unico database** (trasparenza)

Database Distribuito

Definizione

Insieme di database fisicamente distribuiti su diversi dispositivi (nodi) interconnessi tra loro (rete), i cui dati sono inter-relazionati dal punto di vista logico

Dal punto di vista dell'operatore **appare come un unico database** (trasparenza)

Architettura

- **Omogeneo**
 - stesso DBMS (tutti i nodi)
 - stesso schema? (tutti i nodi)
 - minore complessità
- **Eterogeneo**
 - DBMS differenti
 - schemi differenti?
 - maggiore complessità

DatabaseDistribuito

Definizione

Insieme di database fisicamente distribuiti su diversi dispositivi (nodi) interconnessi tra loro (rete), i cui dati sono inter-relazionati dal punto di vista logico

Dal punto di vista dell'operatore **appare come un unico database** (trasparenza)

Architettura

Omogeneo

Eterogeneo

Database Distribuito

Definizione

Insieme di database fisicamente distribuiti su diversi dispositivi (nodi) interconnessi tra loro (rete), i cui dati sono inter-relazionati dal punto di vista logico

Dal punto di vista dell'operatore **appare come un unico database** (trasparenza)

Architettura

Omogeneo

Eterogeneo

Sincronia

- **Sincrono**

- repliche aggiornate immediatamente (tutti i nodi)
- se non possibile l'operazione fallisce (errore)
- stessa risposta interrogando qualunque nodo
- garantite consistenza e coerenza
- minor complessità
- lentezza
- scritture non possibili se connessione assente

- **Asincrono**

- ritardo diffusione modifiche (tra nodi)
- prestazioni
- modifiche possibili anche in assenza di connettività (tra nodi)
- temporanea incoerenza
- complessità maggiore

DatabaseDistribuito

Definizione

Insieme di database fisicamente distribuiti su diversi dispositivi (nodi) interconnessi tra loro (rete), i cui dati sono inter-relazionati dal punto di vista logico

Dal punto di vista dell'operatore **appare come un unico database** (trasparenza)

Architettura

Omogeneo

Eterogeneo

Sincronia

Sincrono

Asincrono

DatabaseDistribuito

Distribuzione

Definizione

Insieme di database fisicamente distribuiti su diversi dispositivi (nodi) interconnessi tra loro (rete), i cui dati sono inter-relazionati dal punto di vista logico

Dal punto di vista dell'operatore **appare come un unico database** (trasparenza)

Architettura

Omogeneo

Eterogeneo

Sincronia

Sincrono

Asincrono

- **Replica**

- distribuzione di copie del DB (su più nodi)
- copie complete o parziali

- **Frammentazione** (partitioning / sharding)

- distribuzione di frammenti di dati (su più nodi)
- **orizzontale** (distribuzione di intere righe)
- **verticale** (distribuzione di colonne)

DatabaseDistribuito

Definizione

Insieme di database fisicamente distribuiti su diversi dispositivi (nodi) interconnessi tra loro (rete), i cui dati sono inter-relazionati dal punto di vista logico

Dal punto di vista dell'operatore **appare come un unico database** (trasparenza)

Architettura

Omogeneo

Eterogeneo

Sincronia

Sincrono

Asincrono

Distribuzione

Replica

Frammentazione

Sincronizzazione

Sincronizzazione

Definizione

Mantenimento di **consistenza**, **coerenza** e **integrità** delle varie repliche sparse all'interno di un sistema

- **Consistenza**

tutti i nodi restituiscono la versione già aggiornata dell'informazione, la modifica si propaga **immediatamente**, se non è possibile l'operazione fallisce (errore)

- **Coerenza**

ogni nodo riceve la versione aggiornata dell'informazione, la modifica si propaga **appena possibile** (temporanea incoerenza)

- **Integrità**

completezza, **accuratezza** e **correttezza** dei dati nel loro complesso anche rispetto a eventuali riferimenti tra un dato e l'altro (integrità referenziale)

Sincronizzazione

Definizione

Mantenimento di consistenza, coerenza e integrità delle varie repliche sparse all'interno di un sistema

Consistenza

Coerenza

Integrità

Definizione

Mantenimento di consistenza, coerenza e integrità delle varie repliche sparse all'interno di un sistema

Consistenza

Coerenza

Integrità

Sincronizzazione

Cosa può servirci?

- **Strategia**
 - una serie di **regole** e **policy** ben definite
 - sincronizza i dati di una copia (dispositivo remoto) con quelli di un'altra (sede centrale)
- **Astrazione?**
 - layer che semplifica la realizzazione dell'applicazione
 - libreria/componenti (iORM)
- **Spazio condiviso?**
 - un DB centrale (sede? cloud?)
 - stato globale del sistema
 - accessibile a tutti i dispositivi

Sincronizzazione

Definizione

Mantenimento di consistenza, coerenza e integrità delle varie repliche sparse all'interno di un sistema

Consistenza

Coerenza

Integrità

Cosa può servirci?

Strategia

Astrazione

Spazio condiviso

Definizione

Mantenimento di consistenza, coerenza e integrità delle varie repliche sparse all'interno di un sistema

Consistenza

Coerenza

Integrità

Cosa può servirci?

Strategia

Astrazione

Spazio condiviso

Sincronizzazione

Requisiti facoltativi

- **Trasparenza**

- interazione utente-dispositivo non appesantita in modo che il lavoro non ne risenta?
- richiesto un atto esplicito per avviare il processo?

- **Consapevolezza**

- **network awareness**: utente conscio se on-line/off-line?
- **synchro awareness**: utente conscio se sincronizzazione in atto?

- **Flessibilità**

- sincronizzazione di una parte delle informazioni (disponibilità)
- in base a permessi e ruoli?
- In base alla finalità dell'app locale? (stato avanzamento cantiere, assistenze, manutenzioni)

Sincronizzazione

Definizione

Mantenimento di consistenza, coerenza e integrità delle varie repliche sparse all'interno di un sistema

Consistenza

Coerenza

Integrità

Cosa può servirci?

Strategia

Astrazione

Spazio condiviso

Requisiti facoltativi

Trasparenza

Consapevolezza

Flessibilità

Sincronizzazione

Ruoli

- **Source**
 - nodo da cui estrarre i dati
- **Target**
 - nodo a cui trasferire i dati
- **OneToMany**
 - i cambiamenti avvenuti in un nodo (source) potrebbero dover essere resi noti a più target

Definizione

Mantenimento di consistenza, coerenza e integrità delle varie repliche sparse all'interno di un sistema

Consistenza Coerenza Integrità

Cosa può servirci?

Strategia Astrazione Spazio condiviso

Requisiti facoltativi

Trasparenza Consapevolezza Flessibilità

Sincronizzazione

Definizione

Mantenimento di consistenza, coerenza e integrità delle varie repliche sparse all'interno di un sistema

Consistenza Coerenza Integrità

Cosa può servirci?

Strategia Astrazione Spazio condiviso

Requisiti facoltativi

Trasparenza Consapevolezza Flessibilità

Ruoli

Source Target OneToMany

Sincronizzazione

Direzione

- **Monodirezionale**
 - da un nodo ad un altro nodo
 - one-way
- **Bidirezionale**
 - da un nodo ad un altro nodo e viceversa
 - due sincronizzazioni monodirezionali in successione nelle quali i due nodi si scambiano i ruoli

Definizione

Mantenimento di consistenza, coerenza e integrità delle varie repliche sparse all'interno di un sistema

Consistenza Coerenza Integrità

Cosa può servirci?

Strategia Astrazione Spazio condiviso

Requisiti facoltativi

Trasparenza Consapevolezza Flessibilità

Ruoli

Source Target OneToMany

Sincronizzazione

Definizione

Mantenimento di consistenza, coerenza e integrità delle varie repliche sparse all'interno di un sistema

Consistenza Coerenza Integrità

Cosa può servirci?

Strategia Astrazione Spazio condiviso

Requisiti facoltativi

Trasparenza Consapevolezza Flessibilità

Ruoli

Source Target OneToMany

Direzione

Monodirezionale Bidirezionale

Sincronizzazione

Entità del flusso

Definizione

Mantenimento di consistenza, coerenza e integrità delle varie repliche sparse all'interno di un sistema

Consistenza Coerenza Integrità

Cosa può servirci?

Strategia Astrazione Spazio condiviso

Requisiti facoltativi

Trasparenza Consapevolezza Flessibilità

Ruoli

Source Target OneToMany

Direzione

Monodirezionale Bidirezionale

- **Full synchronization**

- trasferisce una copia completa dei dati, modificati e non
- **minor complessità**
- **traffico elevato** (overhead infrastruttura di rete)

- **Incremental synchronization**

- solo i dati che hanno subito un aggiornamento (dall'ultima sincronizzazione)
- **efficienza**
- **maggior complessità**
- **individuazione dei dati creati/modificati/eliminati** (dall'ultima sincronizzazione)

- **Differential synchronization** (google docs)

- algoritmo state-based
- ciclo continuo in background di operazioni diff & patch
 - individuazione diff tra due files
 - individuazione delle modifiche (patch) da applicare all'altro file affinché diventi identico
 - propagazione a tutte le altre copie

Sincronizzazione

Definizione

Mantenimento di consistenza, coerenza e integrità delle varie repliche sparse all'interno di un sistema

Consistenza Coerenza Integrità

Cosa può servirci?

Strategia Astrazione Spazio condiviso

Requisiti facoltativi

Trasparenza Consapevolezza Flessibilità

Ruoli

Source Target OneToMany

Direzione

Monodirezionale Bidirezionale

Entità del flusso

Full Incremental Differential

SUMMARY



SUMMARY

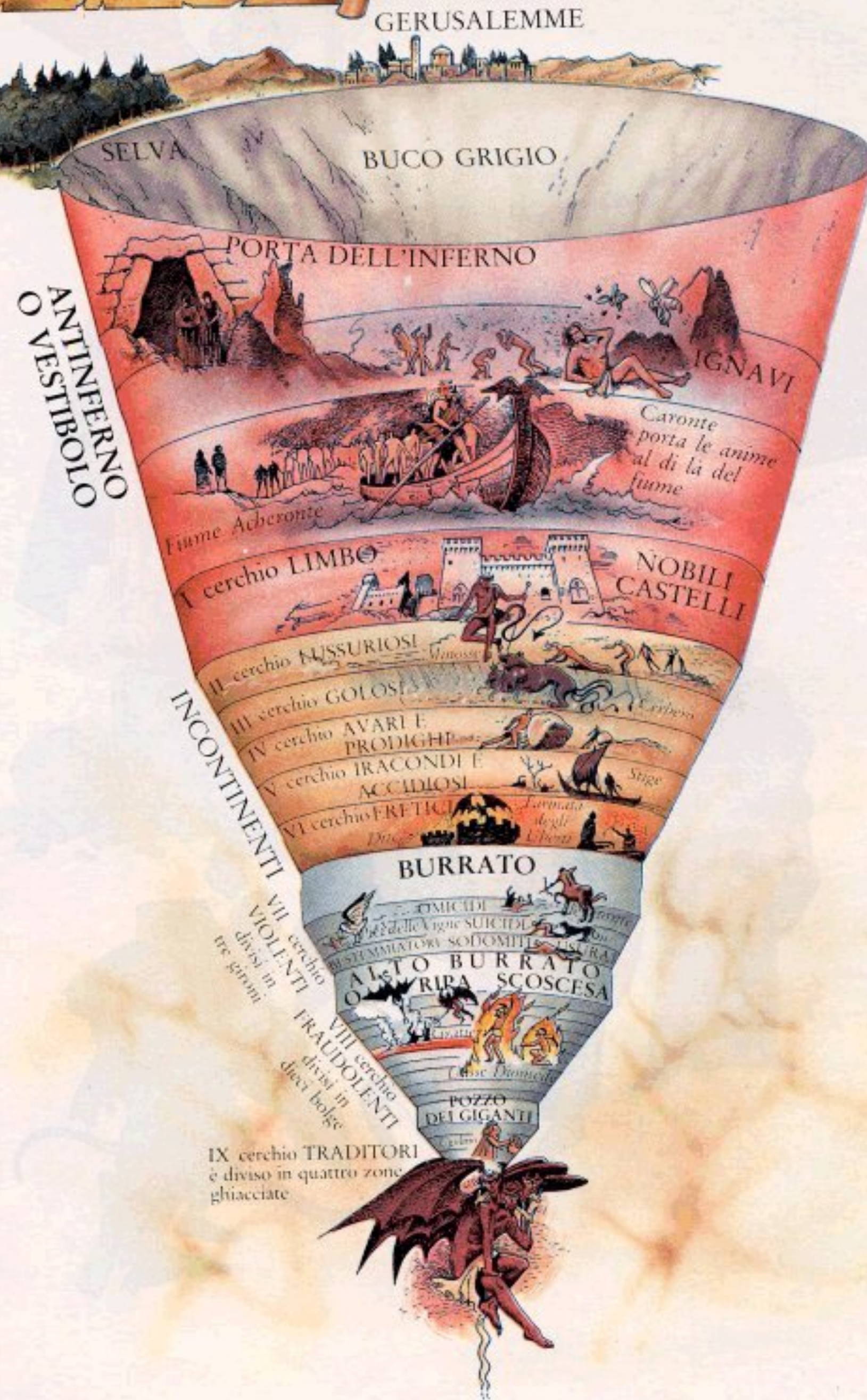
- Avere **repliche**, anche parziali, dei dati sparse per il sistema consente un accesso ai dati “**anytime, anywhere**” anche quando la connessione viene a mancare o non è stabile (aumento disponibilità)
- **mantenimento** di una buona **user-experience** anche in caso di connessione non performante
- **trasparenza** nell’accesso ai dati e nell’interazione con essi, l’operatore non percepisce differenze se **on-line** oppure **off-line**
- **possibili complicazioni !!!**



**KEEP
CALM
AND**

**Lasciate ogni speranza
o voi che entrate!**

FIGURAZIONE GENERALE
DELL'INFERNO

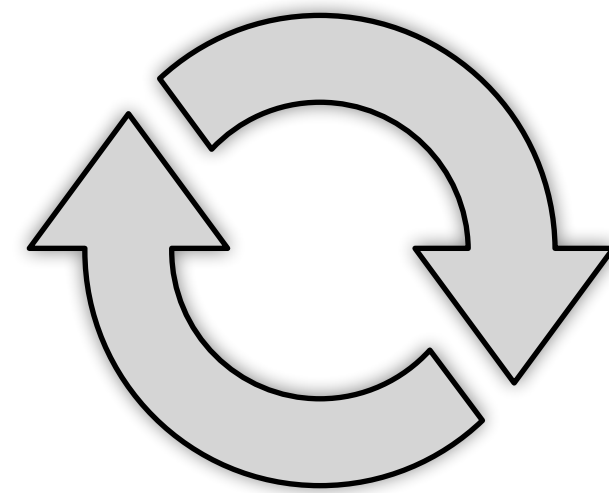


Sincronizzazione

Sincronizzazione



Source

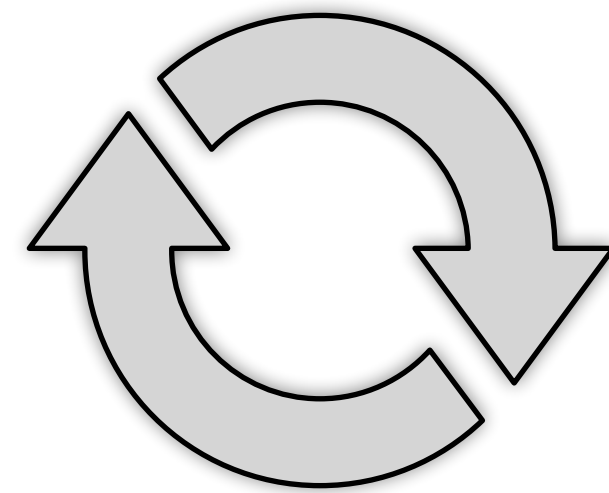


Target

Sincronizzazione

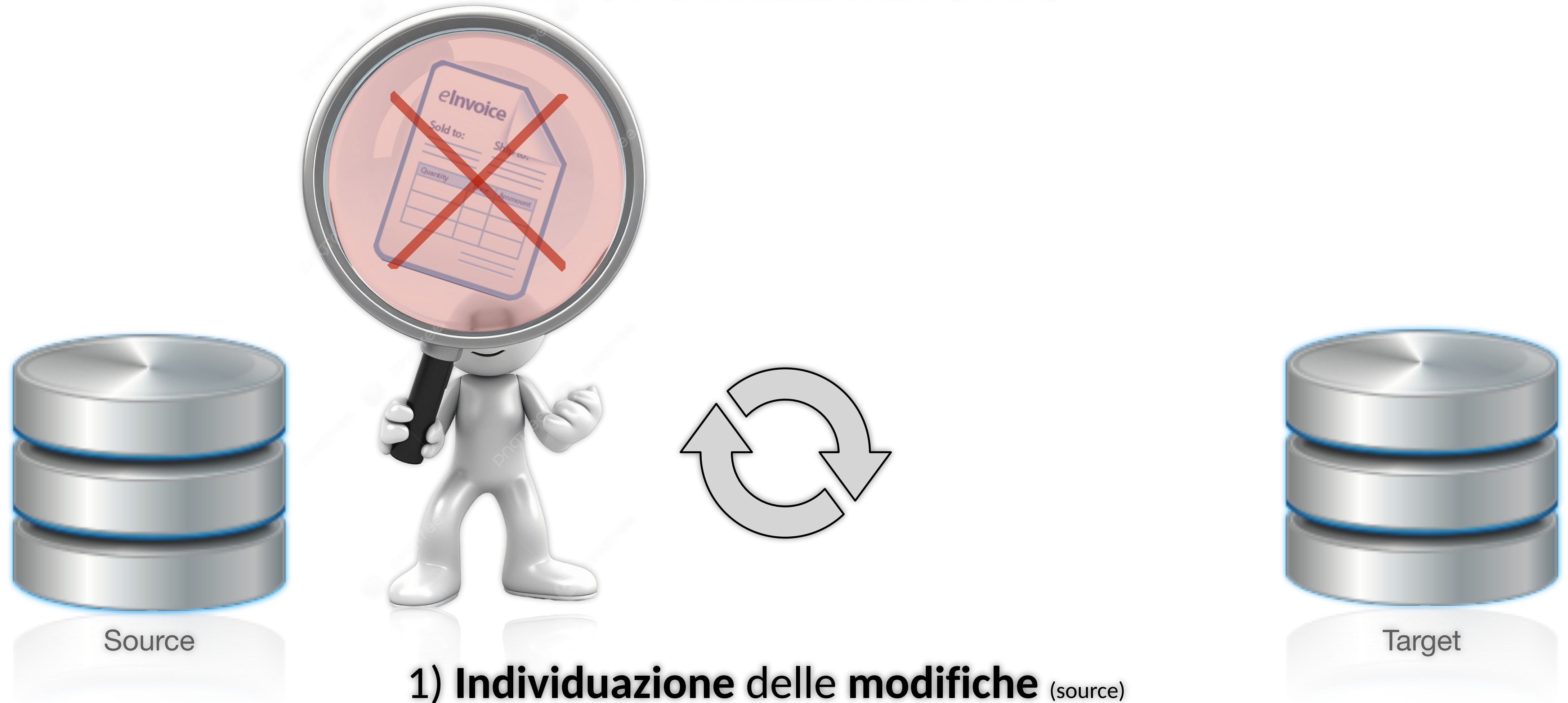


Source



Target

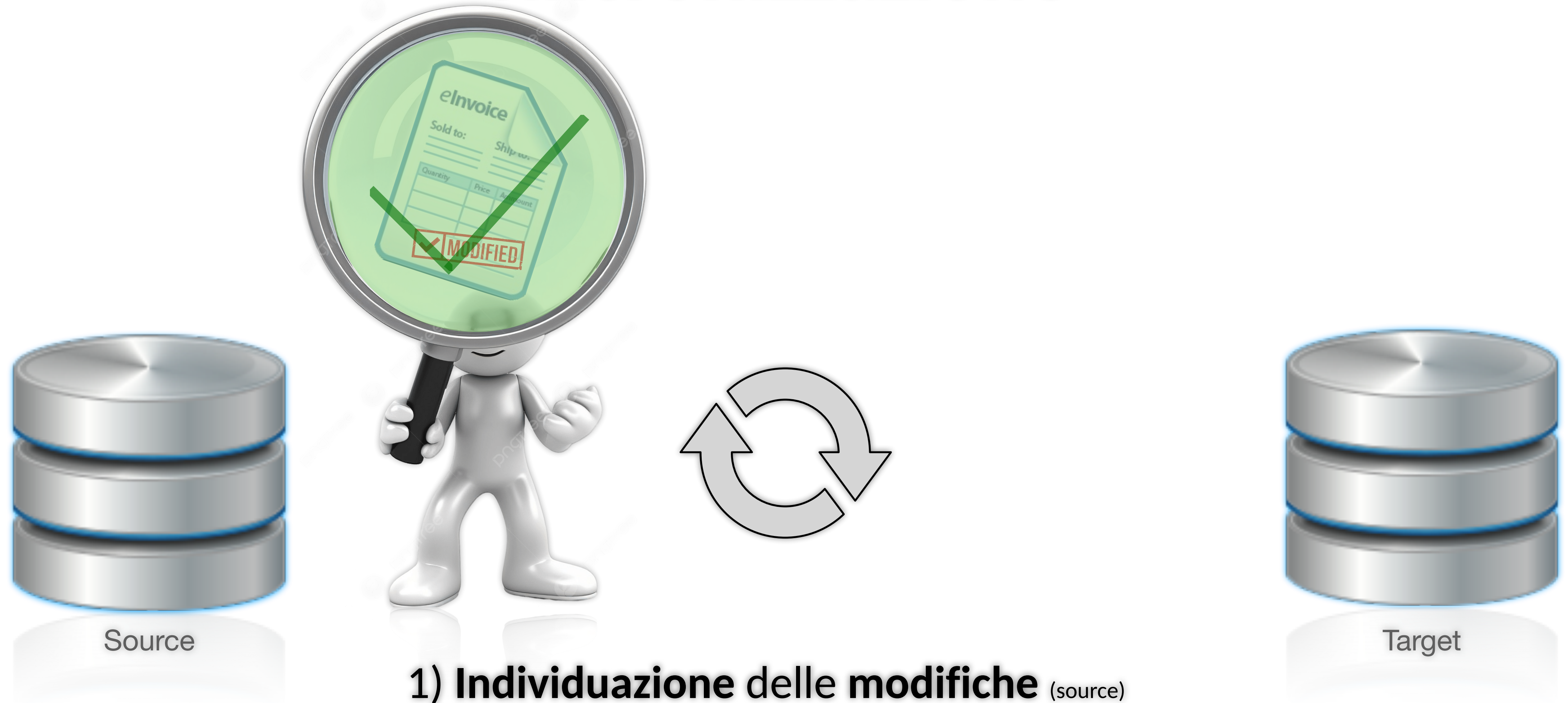
Sincronizzazione



Sincronizzazione



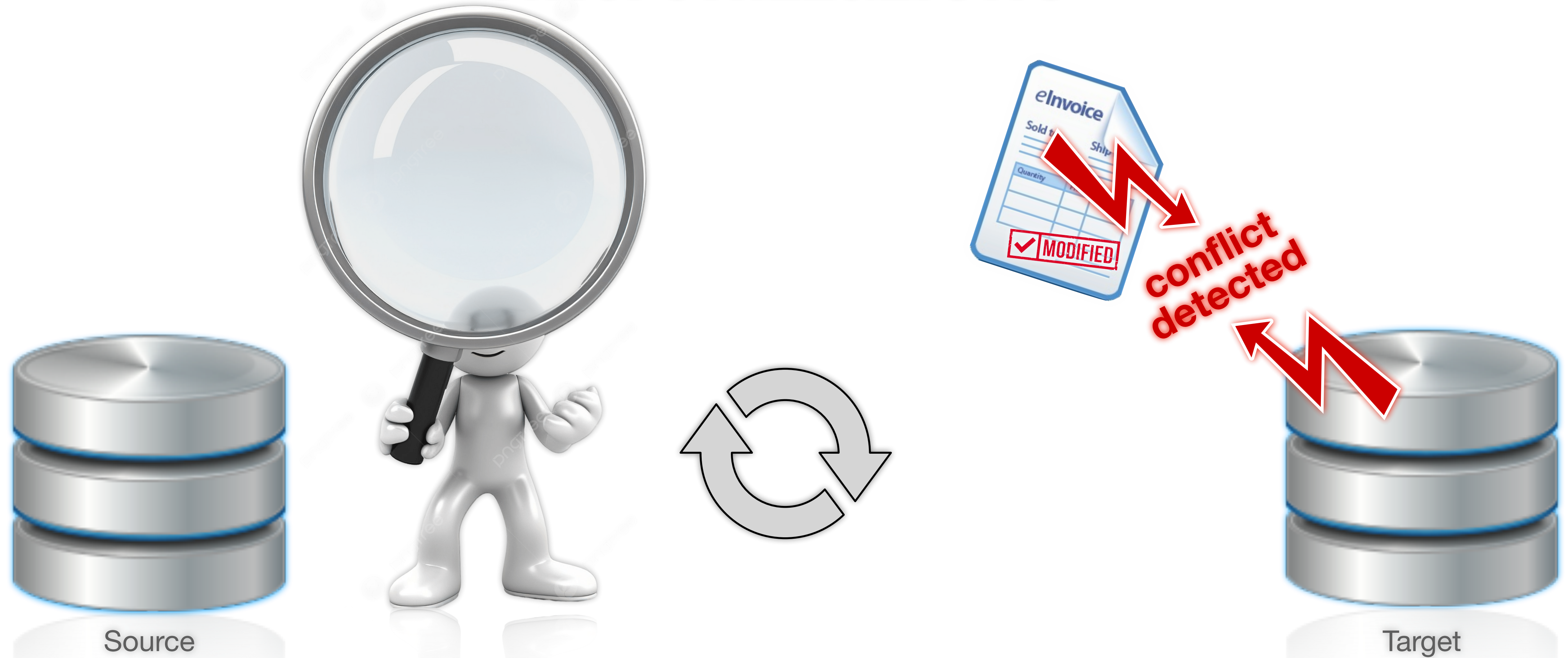
Sincronizzazione



Sincronizzazione



Sincronizzazione



- 1) Individuazione delle modifiche (source)
- 2) Rilevamento dei conflitti

Sincronizzazione



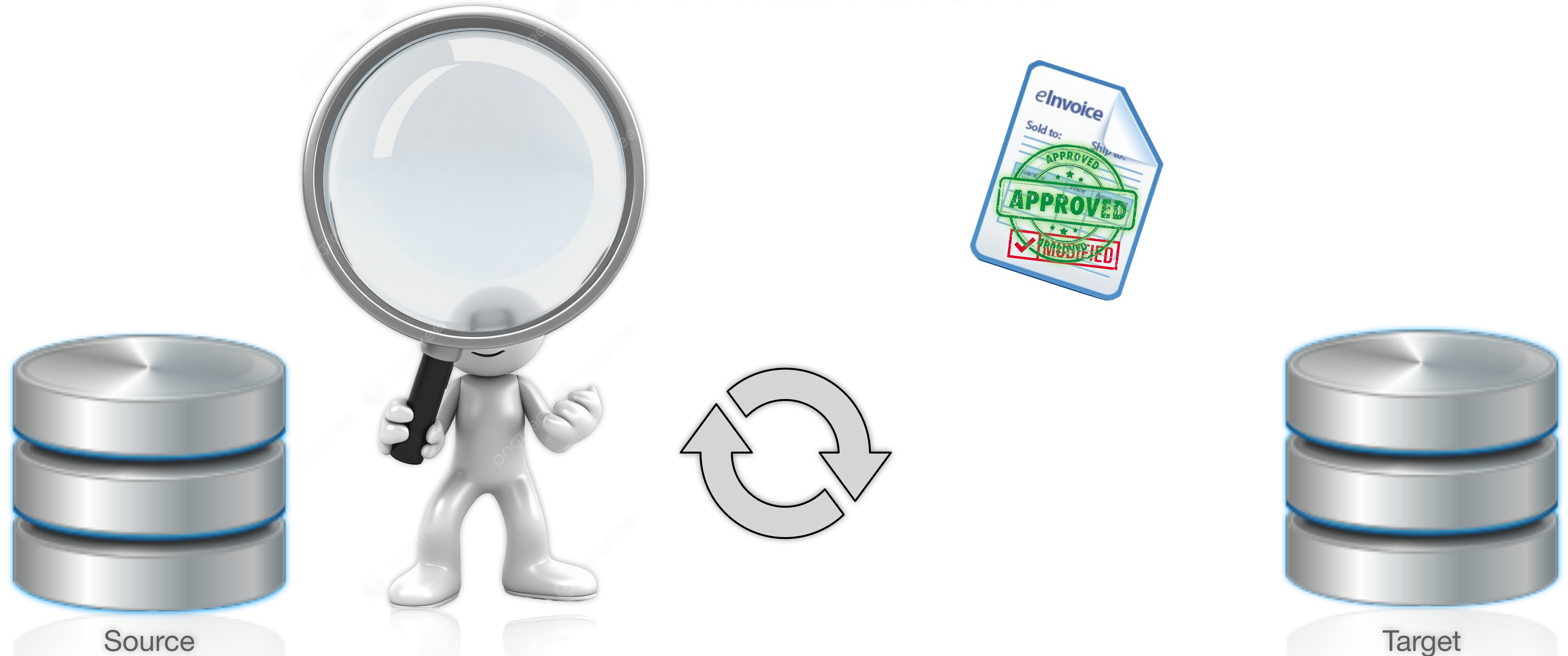
- 1) Individuazione delle modifiche (source)
- 2) Rilevamento dei conflitti
- 3) Gestione/Risoluzione dei conflitti

Sincronizzazione



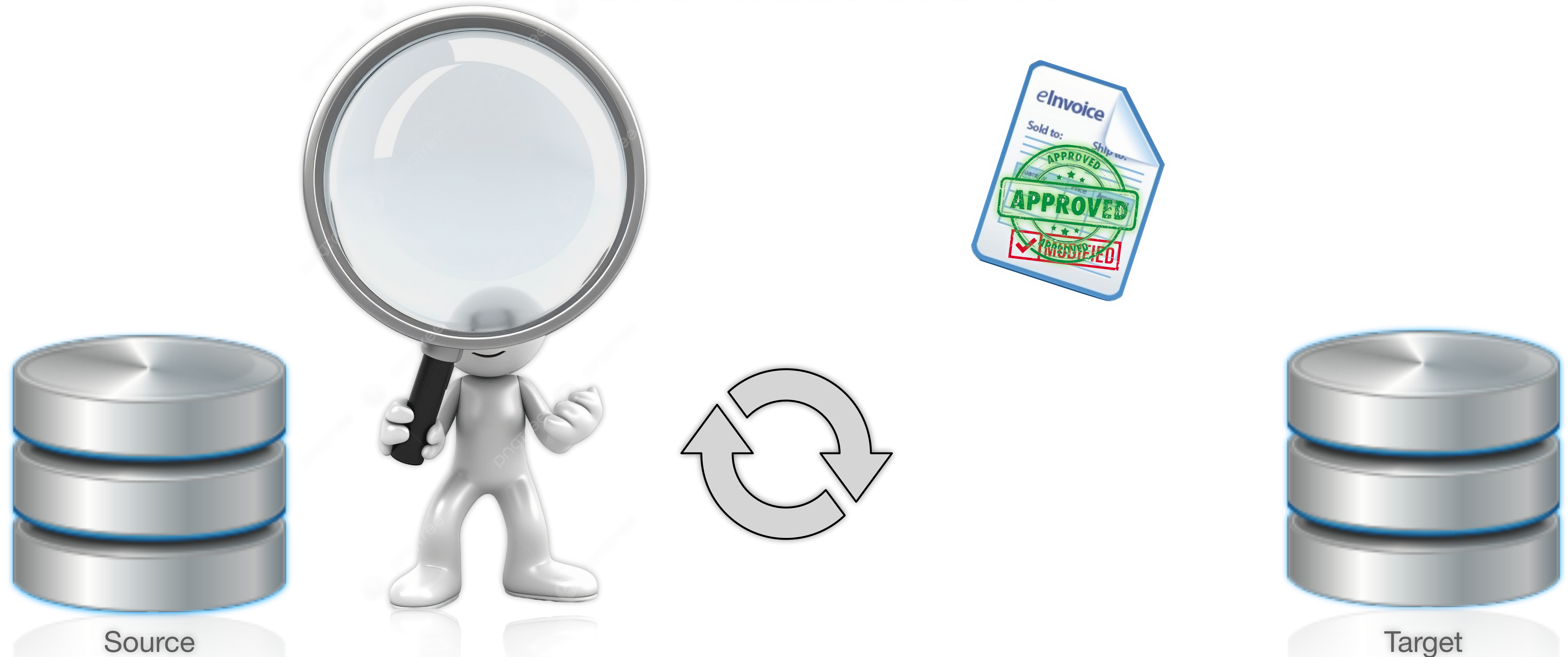
- 1) Individuazione delle modifiche (source)
- 2) Rilevamento dei conflitti
- 3) Gestione/Risoluzione dei conflitti

Sincronizzazione



- 1) Individuazione delle modifiche (source)
- 2) Rilevamento dei conflitti
- 3) Gestione/Risoluzione dei conflitti

Sincronizzazione



- 1) Individuazione delle modifiche (source)
- 2) Rilevamento dei conflitti
- 3) Gestione/Risoluzione dei conflitti
- 4) Persistenza nuova versione (target)

TeoremaCAP

TeoremaCAP

Formulato nel 1998-2000 da Eric Brewer (principio-congettura),
dimostrato nel 2002 da Seth Gilbert e Nancy Lynch (MIT)

Consistency

- tutti i nodi restituiscono l'ultima versione del dato
- ogni modifica deve immediatamente propagarsi su tutti gli altri (per essere considerata riuscita)
- i nodi non raggiungibili devono smettere di rispondere (restituire un errore?)

Availability

- accesso ai dati sempre garantito, su qualsiasi nodo
- tutti i nodi restituiscono una risposta valida, sempre, senza eccezioni
- un nodo non può non rispondere

Partition tolerance

- Il sistema deve essere in grado di funzionare anche in caso di malfunzionamento o indisponibilità di uno o più nodi (es: rottura di un dispositivo, malfunzionamento della rete)

TeoremaCAP

Enunciato

Solo due delle caratteristiche elencate possono essere garantite contemporaneamente

La rimanente NON può essere garantita oppure può esserlo solo in parte

Consistency

Availability

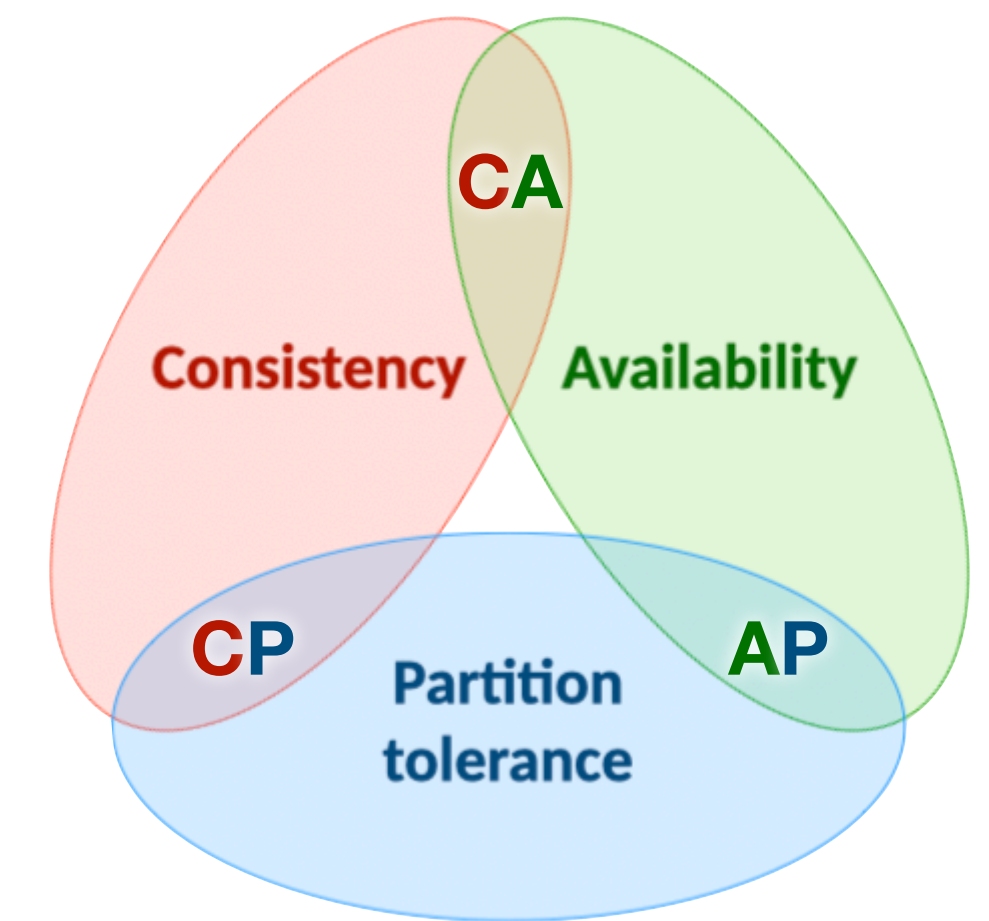
Partition
tolerance

TeoremaCAP

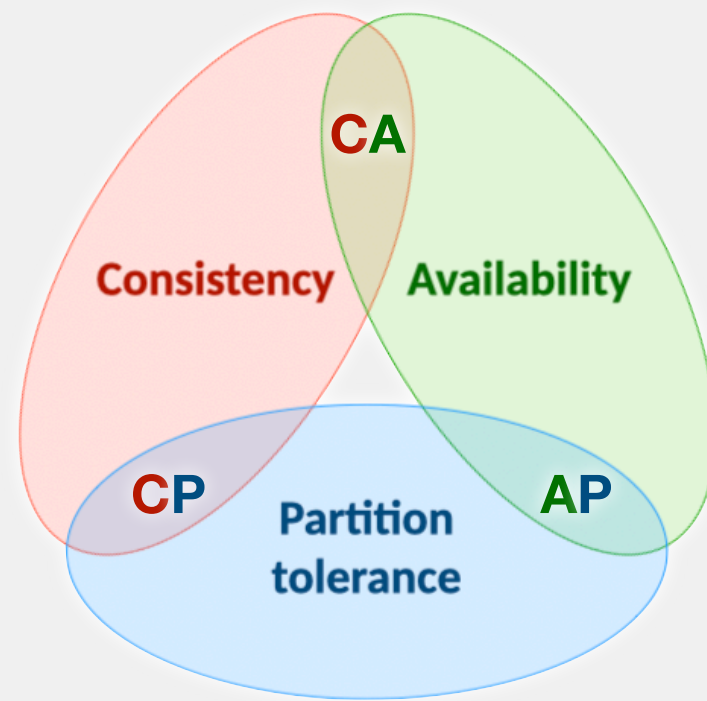
Enunciato

Solo due delle caratteristiche elencate possono essere garantite contemporaneamente

La rimanente NON può essere garantita oppure può esserlo solo in parte



TeoremaCAP



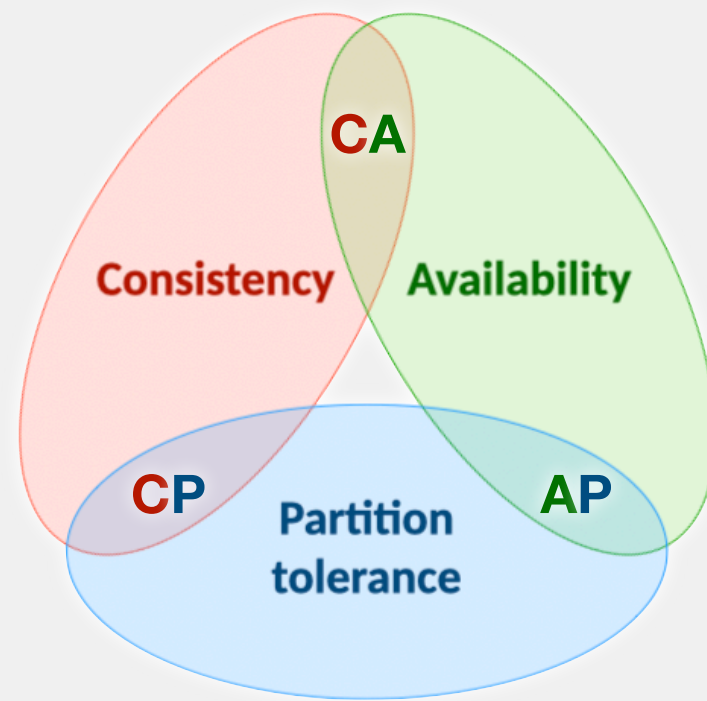
Enunciato

Solo due delle caratteristiche elencate possono essere garantite contemporaneamente

La rimanente NON può essere garantita oppure può esserlo solo in parte



TeoremaCAP



Enunciato

Solo due delle caratteristiche elencate possono essere garantite contemporaneamente

La rimanente NON può essere garantita oppure può esserlo solo in parte

quale caratteristica sacrificare?

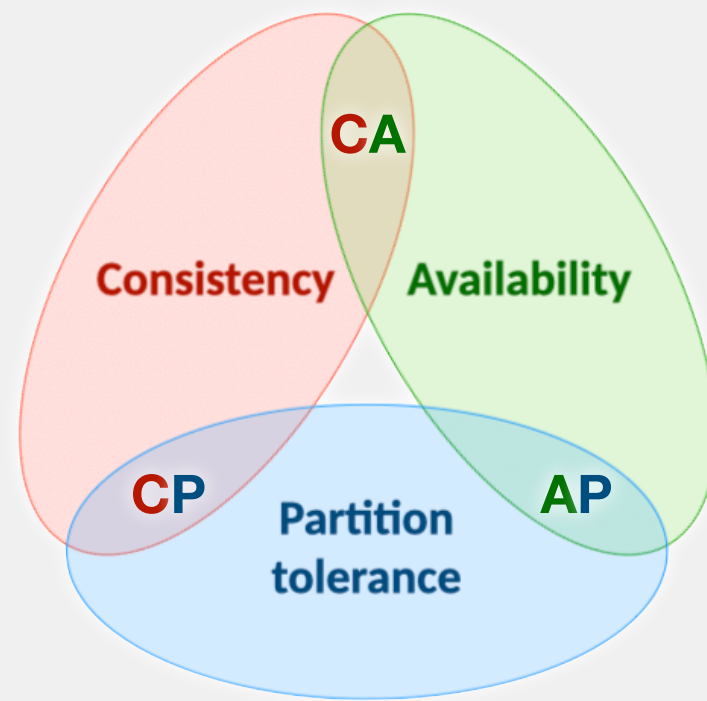
si dovrà tenere conto degli **aspetti tecnici**

ma soprattutto delle **logiche di business**

quali sono le funzionalità che devono sempre essere disponibili anche a costo di lavorare con dati non aggiornati? (es: DNS)

quali sono invece le operazioni che il sistema deve effettuare solo ed esclusivamente con dati consistenti/coerenti? (es: transazioni finanziarie)





Enunciato

Solo due delle caratteristiche elencate possono essere garantite contemporaneamente

La rimanente NON può essere garantita oppure può esserlo solo in parte

TeoremaCAP

Consistency differenze CAP-ACID

importante non confondere il significato di consistenza del teorema CAP con quello previsto dalle caratteristiche ACID delle transazioni di un DB relazionale

- **CAP**

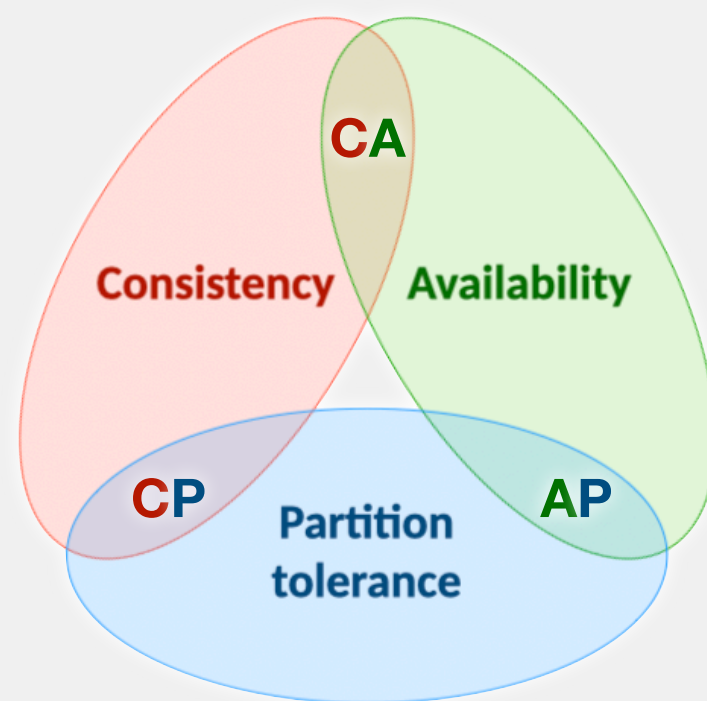
consistenza in termini di **accesso al valore più aggiornato** di un certo dato, quello persistito nell'ultima scrittura dello stesso

- **ACID** (Atomicity, Consistency, Isolation, Durability)

la consistenza fa riferimento al **rispetto dei vincoli di integrità, prima e dopo l'esecuzione di una transazione**

una transazione porta il DB da uno stato consistente ad un nuovo stato consistente

non ci sono mai momenti in cui i dati del database non rispettano i vincoli di integrità



Enunciato

Solo due delle caratteristiche elencate possono essere garantite contemporaneamente

La rimanente NON può essere garantita oppure può esserlo solo in parte

TeoremaCAP

Eventual Consistency transazioni B.A.S.E.

Rinunciando alla consistenza (CAP) è possibile realizzare meccanismi basati su:

- messaggi/notifiche (code, meccanismi public-subscribe)
- sincronizzazioni (periodiche o a richiesta)

i dati locali saranno **sempre** sicuramente **disponibili**

non è detto che, in un dato istante e in un certo nodo, saranno i più aggiornati per via dell'inevitabile **ritardo di propagazione**

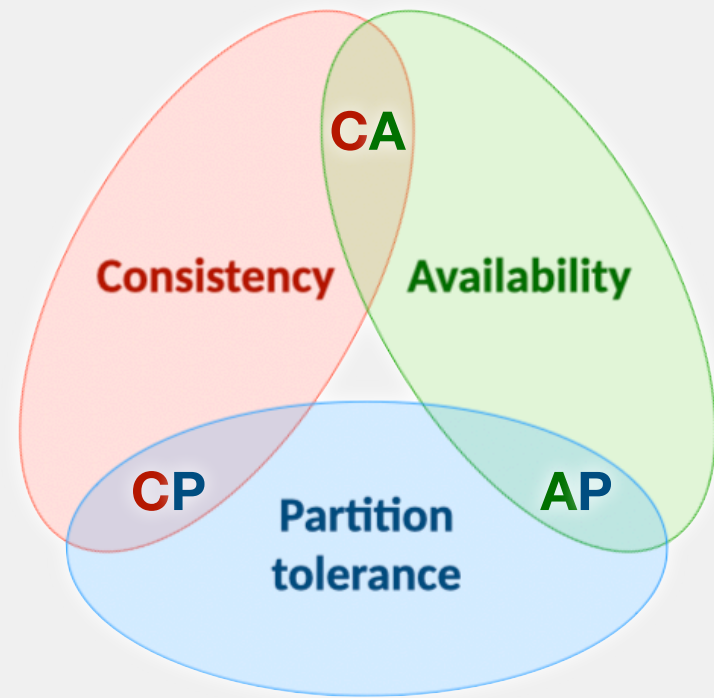
a ben vedere, **non si rinuncia completamente alla consistenza CAP** tra i vari nodi, essi **tenderanno comunque a raggiungerla** anche se **non istantaneamente**

(non si potrà garantire l'istante in cui ciò accadrà)

quindi è tollerata una **temporanea inconsistenza**

in questo caso si parla di **Eventual Consistency** e di modello transazionale **BASE**
(Basically, Available, Soft state, Eventually consistent)

TeoremaCAP



Enunciato

Solo due delle caratteristiche elencate possono essere garantite contemporaneamente

La rimanente NON può essere garantita oppure può esserlo solo in parte

Eventual Consistency

dati locali sempre disponibili

ritardo di propagazione

temporanea inconsistenza

Concorrenza

Concorrenza

- E' possibile che due operatori modifichino **contemporaneamente** lo **stesso dato** mentre sono disconnessi (copia locale)
- Possibile **perdita** di **consistenza**
- Garantire la possibilità a più utenti di effettuare **concorrentemente** modifiche sulla stesso dato è complicato
- Necessario introdurre “**meccanismi**” per il **controllo della concorrenza**



Concorrenza

controllo della concorrenza



- **Locking pessimistico**

- Prima di poter modificare un dato è necessario ottenere un “Lock” che dovrà essere rilasciato al termine dell’operazione
- Serializzazione delle modifiche
- Semplicità
- Garanzia di coerenza, consistenza e integrità
- Impossibile se off-line
- Possibili deadlock

- **Locking ottimistico**

- Nessun blocco a inizio operazioni di modifica
- Controllo violazioni di integrità e consistenza a termine operazione
- In caso di conflitto servono regole e policy (strategia), manuali o automatiche, per la sua risoluzione (rollback e riesecuzione?)
- Possibile perdita di informazioni !!!



Concorrenza

controllo della concorrenza

Locking pessimistico

Locking ottimistico

Concorrenza

individuare i dati da sincronizzare



- **Flag**

- Semplicità
- Solo monodirezionale (client → server)

- **Timestamp**

- Istante dell'ultima modifica di un dato
- Se last update > data ultima sincronizzazione (salvo da qualche parte la data ultima sincro)
- Non serve tenere traccia delle precedenti modifiche e/o sincronizzazioni
- Tempo sincronizzato per tutti i dispositivi

- **Digest**

- Hash differente = updated (confronto con hash sul server o record con 2 hash: attuale + ultima sincro)

- **Log** (VCS?)

- Memorizzazione cronologica di ogni cambiamento (salvo da qualche parte l'ultimo evento sincronizzato)
- Trasferisco le variazioni non ancora applicate al target (commit/timeslot)

controllo della concorrenza

Locking pessimistico

Locking ottimistico

Concorrenza

individuare i dati da sincronizzare



controllo della concorrenza

Locking pessimistico

Locking ottimistico



- **Flag**

- Semplicità
- Solo monodirezionale (client → server)



- **Timestamp**

- Istante dell'ultima modifica di un dato
- Se last update > data ultima sincronizzazione (salvo da qualche parte la data ultima sincro)
- Non serve tenere traccia delle precedenti modifiche e/o sincronizzazioni
- Tempo sincronizzato per tutti i dispositivi

- ~~Digest~~

- Differente = updated (confronto con hash sul server o record con 2 hash: attuale + ultima sincro)



- **Log**

- Memorizzazione cronologica di ogni cambiamento (salvo da qualche parte l'ultimo evento sincronizzato)
- Trasferimento delle variazioni non ancora applicate al target (commit/timeslot)



Concorrenza

individuare i dati da sincronizzare



controllo della concorrenza

Locking pessimistico

Locking ottimistico



- **Flag**

- Semplicità
- Solo monodirezionale (client → server)



- **Timestamp**

- Istante dell'ultima modifica di un dato
- Se last update > data ultima sincronizzazione (salvo da qualche parte la data ultima sincro)
- Non serve tenere traccia delle precedenti modifiche e/o sincronizzazioni
- Tempo sincronizzato per tutti i dispositivi



- ~~Digest~~

- Hash differenziale (confronto con hash sul server o record con 2 hash: attuale + ultima sincro)



- **Log** (VCS?)

- Memorizzazione logica di ogni cambiamento (salvo da qualche parte l'ultimo evento sincronizzato)
- Trasferisco le variazioni non ancora applicate al target (commit/timeslot)

Concorrenza

controllo della concorrenza

Locking pessimistico

Locking ottimistico

individuare i dati da sincronizzare

Flag

Timestamp

Digest

Log

Concorrenza

ID nuovi oggetti (client-side)



- **GUID**

- Non ordinabile
- Univocità non garantita al 100%

- **Device ID based**

- Device ID univoco per ogni dispositivo

- ~~Compartimentazione~~

- Dispositivo 1 = ID da 1 a 1000; dispositivo 2 = ID da 1001 a 2000...

- **Chiave composta**

- MacAddress univoco per ogni dispositivo

- **ID temporaneo** (sarà sostituito durante la sincronizzazione)

- Complessità



controllo della concorrenza

Locking pessimistico

Locking ottimistico

individuare i dati da sincronizzare

Flag

Timestamp

Digest

Log

Concorrenza

controllo della concorrenza

Locking pessimistico

Locking ottimistico

individuare i dati da sincronizzare

Flag

Timestamp

Digest

Log

ID nuovi oggetti (client-side)

GUID

Device ID based

Compartim

Chiave composta

ID temporaneo

Concorrenza

rilevamento dei conflitti

- **Controllo della versione** (versionamento)

- Numero

Se versione server > versione mantenuta sul client = conflitto

NB: sul client remoto non si aggiorna

- Timestamp

Se last update server > last update mantenuta sul client = conflitto

NB: sul client remoto non si aggiorna

- Qualsiasi tipo di dato ordinabile (cronologia)

- **Digest**

- Hash

Se hash calcolato al momento <> hash sul server

NB: sul client remoto salvo anche l'hash del server



**conflict
detected**

controllo della concorrenza

Locking pessimistico

Locking ottimistico

individuare i dati da sincronizzare

Flag

Timestamp

Digest

Log

ID nuovi oggetti (client-side)

GUID

Device ID based

Compartim

Chiave composta

ID temporaneo

Concorrenza

controllo della concorrenza

Locking pessimistico

Locking ottimistico

individuare i dati da sincronizzare

Flag

Timestamp

Digest

Log

ID nuovi oggetti (client-side)

GUID

Device ID based

Compartim

Chiave composta

ID temporaneo

rilevamento dei conflitti

Controllo della versione

Digest

Concorrenza

gestione dei conflitti

regole e policy per la risoluzione dei conflitti



- **Conflict strategies**

- Same version win
- Last update win
- Longest history win
- Merge changes
- Manual (rollback, reload, edit again)

- **Multiversion Concurrency Control** (MCC / MVCC)

- Alla modifica del dato **aggiunge la nuova versione ma non elimina la vecchia**
- La **vecchia versione** viene marcata come **obsoleta**
- **Più versioni** dello stesso dato
- Finché scrittura nuova versione non completa sarà fornita la precedente (disponibilità)
- In caso di conflitto solo una delle due versioni verrà scelta (possibile perdita di informazioni), quella scartata sarà comunque mantenuta per eventuale ripristino

controllo della concorrenza

Locking pessimistico

Locking ottimistico

individuare i dati da sincronizzare

Flag

Timestamp

Digest

Log

ID nuovi oggetti (client-side)

GUID

Device ID based

Compartim

Chiave composta

ID temporaneo

rilevamento dei conflitti

Controllo della versione

Digest

Concorrenza

controllo della concorrenza

Locking pessimistico

Locking ottimistico

individuare i dati da sincronizzare

Flag

Timestamp

Digest

Log

ID nuovi oggetti (client-side)

GUID

Device ID based

Compartim

Chiave composta

ID temporaneo

rilevamento dei conflitti

Controllo della versione

Digest

rilevamento dei conflitti

Conflict strategies

Multiversion Concurrency Control

SUMMARY



SUMMARY

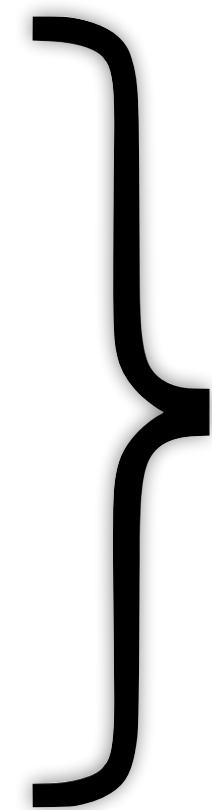
- Dati replicati + connessione non garantita → **locking ottimistico**
- Locking ottimistico = possibile
- Locking ottimistico = possibile
- Va gestito
- **Non locking ottimistico, anche senza replica**

theiORMway



theiORMway

- 1) Versioning
- 2) E.T.M. (Entity Time Machine)
- 3) Conflict strategies
- 4) Synchronization strategies



Concurrency **C**ontrol (MCC / MVCC)



EntityTimeMachine

an iORM feature



EntityTimeMachine

an iORM feature

Chi ha fatto cosa? ...quando?



EntityTimeMachine

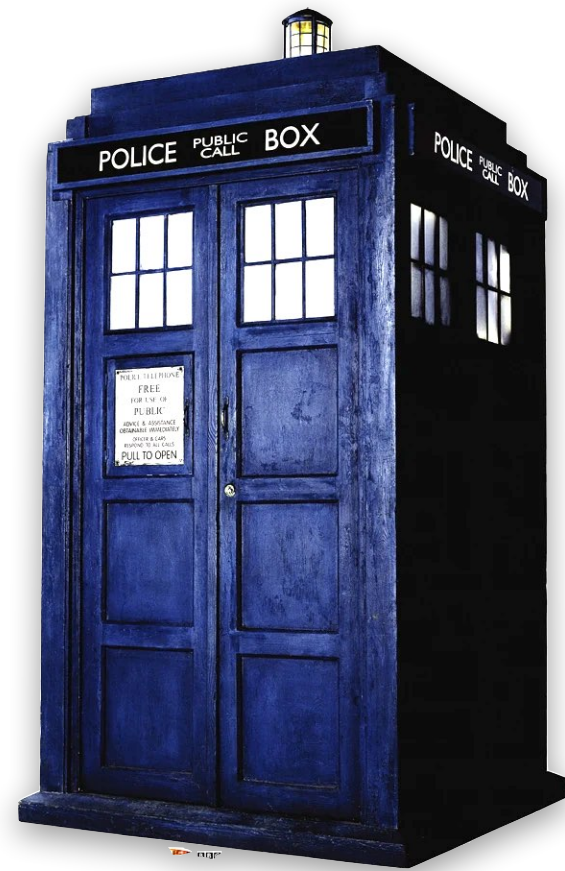
anORMfeature

- **Tracciare** i cambiamenti degli oggetti nel tempo
- **Differenze** tra versioni diverse della stessa entità
- Un oggetto è realmente esistito?
 - ...chi lo ha eliminato?
 - ...quando?
 - ...che **stato** aveva quando è avvenuto il fattaccio?
- **Ripristinare** una versione precedente



EntityTimeMachine

anORMfeature



ETM

- E' un sistema di controllo versione per **oggetti/entità**.
- Archiviare **oggetti** in un repository.
- Creare, modificare, eliminare **entità** tenendo traccia di tutto ciò che accade.
- Vedere **chi** ha compiuto tali azioni e **quando**.
- Rilevare e risolvere **conflitti**.
- **Ripristinare** versioni precedenti quando necessario.



VCS (Version Control System)

- E' un sistema di controllo versione del **codice sorgente**.
- Archiviare **file** in un repository.
- Creare, modificare, eliminare **files** tenendo traccia di tutto ciò che accade.
- Vedere **chi** ha compiuto tali azioni e **quando**.
- Rilevare e risolvere **conflitti**.
- **Ripristinare** versioni precedenti quando necessario.



Creare un repository



Create unrepository



```
unit MyRepository;
```

```
interface
```

```
uses  
  iORM;
```

```
type
```

```
[etmRepository('DBTableName')]
```

```
TEtmTimeSlot = class(TioEtmCustomTimeSlot)
```

```
end;
```

```
implementation
```

```
end.
```

```
etmRepository = ioEntity;  
// Map the class as an entity to be persisted  
ioEntity = class(TioCustomAttribute)  
public  
  constructor Create(const ATableName: String; ...);  
  ...  
end;
```

```
unit iORM.Attributes;  
  
TioEtmCustomTimeSlot = class  
  ...  
public  
  property ID;  
  property DateAndTime;  
  property EventType;  
  property EntityClassName;  
  property EntityID;  
  property EntityVersion;  
  property RevertedFromVersion;  
  property EntityState;  
  property RemoteEntityState;  
  property ConflictType;  
  property UserName;  
  property UserID;  
  ...  
end;
```





Create unrepository



```
unit MyRepository;
```

```
interface
```

```
uses  
  iORM;
```

```
type
```

```
  [etmRepository('DBTableName')]
```

```
  TEtmTimeSlot = class(TioEtmCustomTimeSlot)
```

```
  end;
```

```
implementation
```

```
end.
```

```
etmRepository = ioEntity;  
  
// Map the class as an entity to be persisted  
ioEntity = class(TioCustomAttribute)  
public  
  constructor Create(const ATableName: String; ...);  
  ...  
end;
```

```
unit iORM.Attributes;  
  
TioEtmCustomTimeSlot = class  
  ...  
public  
  property ID;  
  property DateAndTime;  
  property EventType;  
  property EntityClassName;  
  property EntityID;  
  property EntityVersion;  
  property RevertedFromVersion;  
  property EntityState;  
  property RemoteEntityState;  
  property ConflictType;  
  property UserName;  
  property UserID;  
  ...  
end;
```



Creare un repository



```
unit MyRepository;
```

```
interface
```

```
uses  
  iORM;
```

```
type
```

```
  [etmRepository('DBTableName')]
```

```
  TEtmTimeSlot = class(TioEtmCustomTimeSlot)
```

```
  end;
```

```
implementation
```

```
end.
```

```
etmRepository = ioEntity;  
  
// Map the class as an entity to be persisted  
ioEntity = class(TioCustomAttribute)  
public  
  constructor Create(const ATableName: String; ...);  
  ...  
end;
```

```
unit iORM.Attributes;  
  
TioEtmCustomTimeSlot = class  
  ...  
public  
  property ID;  
  property DateAndTime;  
  property EventType;  
  property EntityClassName;  
  property EntityID;  
  property EntityVersion;  
  property RevertedFromVersion;  
  property EntityState;  
  property RemoteEntityState;  
  property ConflictType;  
  property UserName;  
  property UserID;  
  ...  
end;
```




Createunrepository



```
unit MyRepository;
```

```
interface
```

```
uses  
  iORM;
```

```
type
```

```
[etmRepository('DBTableName')]  
[ioConnection('ConnectionName')]  
TEtmTimeSlot = class(TioEtmCustomTimeSlot)  
  
end;
```

```
implementation
```

```
end.
```

```
etmRepository = ioEntity;  
  
// Map the class as an entity to be persisted  
ioEntity = class(TioCustomAttribute)  
public  
  constructor Create(const ATableName: String; ...);  
  ...  
end;
```

Connection where the repository will be persisted

```
unit iORM.Attributes;  
  
TioEtmCustomTimeSlot = class  
...  
public  
  property ID;  
  property DateAndTime;  
  property EventType;  
  property EntityClassName;  
  property EntityID;  
  property EntityVersion;  
  property RevertedFromVersion;  
  property EntityState;  
  property RemoteEntityState;  
  property ConflictType;  
  property UserName;  
  property UserID;  
  ...  
end;
```



Tracciare una classe



Tracciare una classe



```
[ioEntity('CUSTOMERS')]  
TCustomer = class  
private  
  FID: Integer;  
  FName: String;  
  FCity: String;  
  FAddress: String;  
  FPhoneNumber: String;  
  FObjVersion: TioObjVersion;  
  function GetFullAddress: String;  
public  
  property ID: Integer read FID write FID;  
  property Name: String read FName write FName;  
  property City: String read FCity write FCity;  
  property Address: String read FAddress write FAddress;  
  property PhoneNumber: String read FPhoneNumber write FPhoneNumber;  
  property FullAddress: String read GetFullAddress;  
end;
```




Tracciare una classe

```
[ioEntity('CUSTOMERS')]  
[etmTrace(TMyRepository)]
```

```
TCustomer = class  
private
```

```
FID: Integer;
```

```
FName: String;
```

```
FCity: String;
```

```
FAddress: String;
```

```
FPhoneNumber: String;
```

```
FObjVersion: TioObjVersion;
```

```
function GetFullAddress: String;
```

```
public
```

```
property ID: Integer read FID write FID;
```

```
property Name: String read FName write FName;
```

```
property City: String read FCity write FCity;
```

```
property Address: String read FAddress write FAddress;
```

```
property PhoneNumber: String read FPhoneNumber write FPhoneNumber;
```

```
property FullAddress: String read GetFullAddress;
```

```
end;
```

```
etmTrace = class(TCustomAttribute)
```

```
public
```

```
constructor Create(const ATimeSlotClass: TioEtmCustomTimeSlotRef; ...);
```

```
...
```

```
end;
```

ObjVersion is mandatory if you want to use ETM



CRUD



CREATE



READ



UPDATE



DELETE

NOTHING



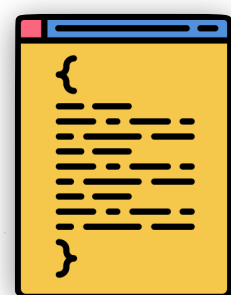
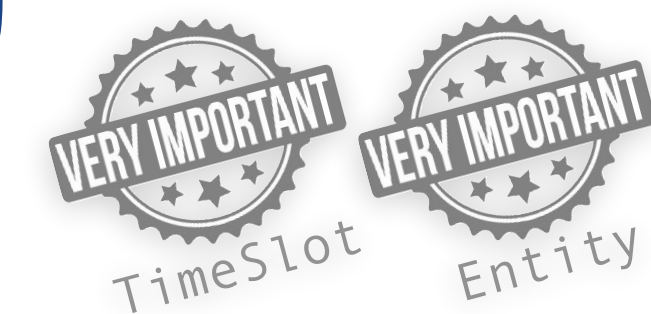
Timeline

o—o—o—o→



Timeline

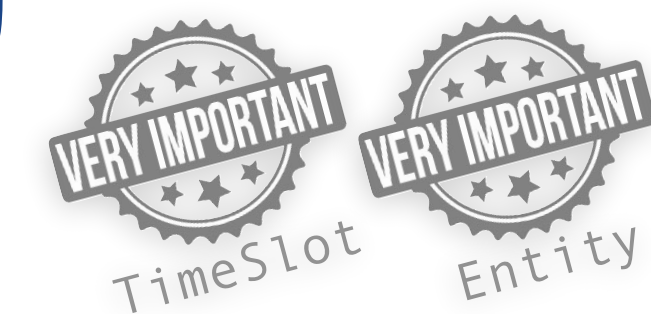
o-o-o-o→



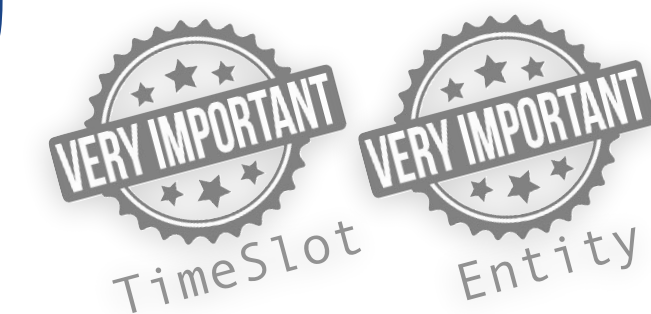
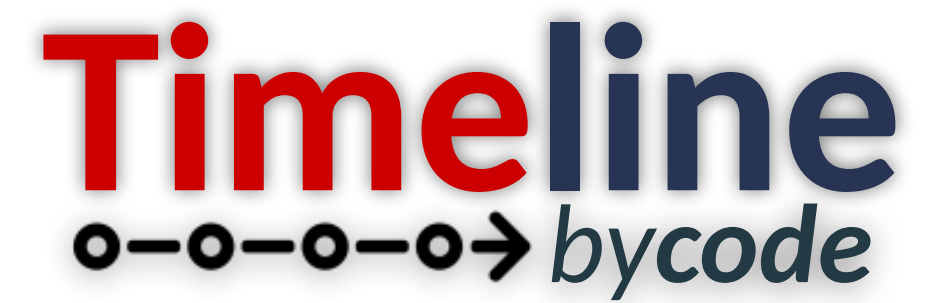
bycode



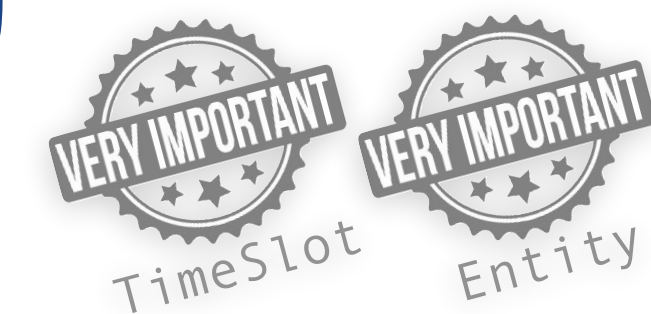
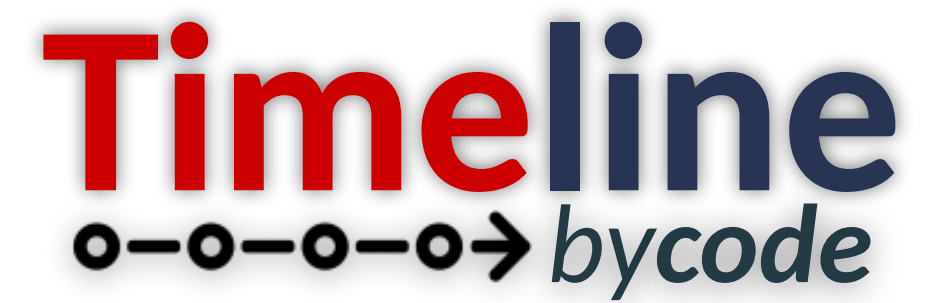
bycomponents



```
var
  LPerson: TPerson;
  LTimeline: TioEtmTimeline;
begin
  ...
  LTimeline := io.etm.TimelineFor(LPerson);
  ...
end;
```



```
var
  LPerson: TPerson;
  LTimeline: ioEtmTimeline;
begin
  ...
  LTimeline := io.etm.TimelineFor(LPerson);
  ...
end;
```

```
var
  LPerson: IPerson;
  LTimeline: TioEtmTimeline;
begin
  ...
  LTimeline := io.etm.TimelineFor(LPerson);
  ...
end;
```



Timeline

o-o-o-o→bycode



```
unit iORM.Attributes;  
...  
// A Timeline is a list of time slots  
TioEtmTimeline = TObjectList<TioEtmCustomTimeSlot>;  
...
```

```
var  
  LPerson: IPerson;  
  LTimeline: TioEtmTimeline;  
begin  
  ...  
  LTimeline := io.etm.TimelineFor(LPerson);  
  ...  
end;
```

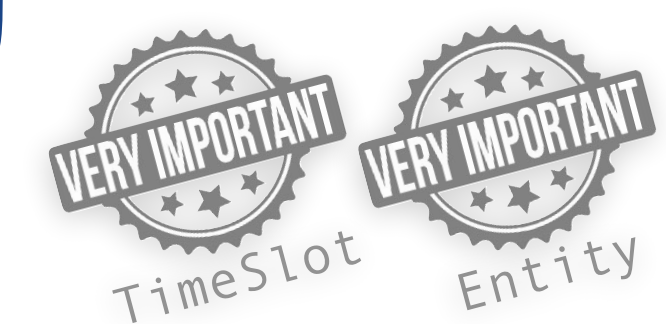
```
unit iORM.Attributes;  
  
TioEtmCustomTimeSlot = class  
...  
public  
  property ID;  
  property DateAndTime;  
  property EventType;  
  property EntityClassName;  
  property EntityID;  
  property EntityVersion;  
  property RevertedFromVersion;  
  property EntityState;  
  property RemoteEntityState;  
  property ConflictType;  
  property UserName;  
  property UserID;  
...
```

```
etmRepository = ioEntity;
```



Timeline

o-o-o-o->bycode



```
unit iORM.Attributes;  
...  
// A Timeline is a list of time slots  
TioEtmTimeline = TObjectList<TioEtmCustomTimeSlot>;  
...
```

```
var  
  LPerson: IPerson;  
  LTimeline: TioEtmTimeline;  
begin  
  ...  
  LTimeline := io.etm.TimelineFor(LPerson);  
end;
```

```
unit iORM.Attributes;  
  
TioEtmCustomTimeSlot = class  
...  
public  
  property ID;  
  property DateAndTime;  
  property EventType;  
  property EntityClassName;  
  property EntityID;  
  property EntityVersion;  
  property RevertedFromVersion;  
  property EntityState;  
  property RemoteEntityState;  
  property ConflictType;  
  property UserName;  
  property UserID;  
...  
end;
```

```
etmRepository = ioEntity;
```




Timeline

o-o-o-o→bycode



```
unit iORM.Attributes;  
...  
// A Timeline is a list of time slots  
TioEtmTimeline = TObjectList<TioEtmCustomTimeSlot>;  
...
```

```
var  
  LPerson: IPerson;  
  LTimeline: TioEtmTimeline;  
  LWhere: IioWhere;  
begin  
  ...  
  LWhere := io.Where('UserName', coEquals, 'Maurizio Del Magno');  
  ...  
  LTimeline := io.etm.TimelineFor(LPerson, LWhere);  
  ...  
end;
```

```
unit iORM.Attributes;  
  
TioEtmCustomTimeSlot = class  
...  
public  
  property ID;  
  property DateAndTime;  
  property EventType;  
  property EntityClassName;  
  property EntityID;  
  property EntityVersion;  
  property RevertedFromVersion;  
  property EntityState;  
  property RemoteEntityState;  
  property ConflictType;  
  property UserName;  
  property UserID;  
...  
end;
```

```
etmRepository = ioEntity;
```



Timeline

o-o-o-o→bycode



```
unit iORM.Attributes;  
...  
// A TimeLine is a list of time slots  
TioEtmTimeline = TObjectList<TioEtmCustomTimeSlot>;  
...
```

```
var  
  LPerson: IPerson;  
  LTimeline: TioEtmTimeline;  
  LWhere: IioWhere;  
begin  
  ...  
  LWhere := io.Where('UserName', coEquals, 'Maurizio Del Magno');  
  ...  
  LTimeline := io.etm.TimelineFor(LPerson, LWhere);  
  ...  
end;
```

// Binding ???

```
ADataset.SetDataObject(LTimeline); // VCL project  
ABindSource.SetDataObject(LTimeline); // FMX project  
AModelPresenter.SetDataObject(LTimeline); // MVVM project
```

```
unit iORM.Attributes;  
  
TioEtmCustomTimeSlot = class  
...  
public  
  property ID;  
  property DateAndTime;  
  property EventType;  
  property EntityClassName;  
  property EntityID;  
  property EntityVersion;  
  property RevertedFromVersion;  
  property EntityState;  
  property RemoteEntityState;  
  property ConflictType;  
  property UserName;  
  property UserID;  
...  
end;
```

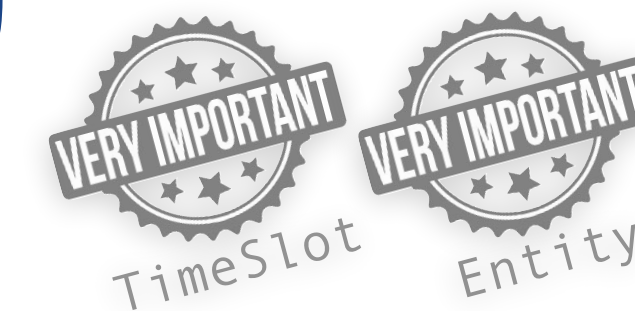
```
etmRepository = ioEntity;
```



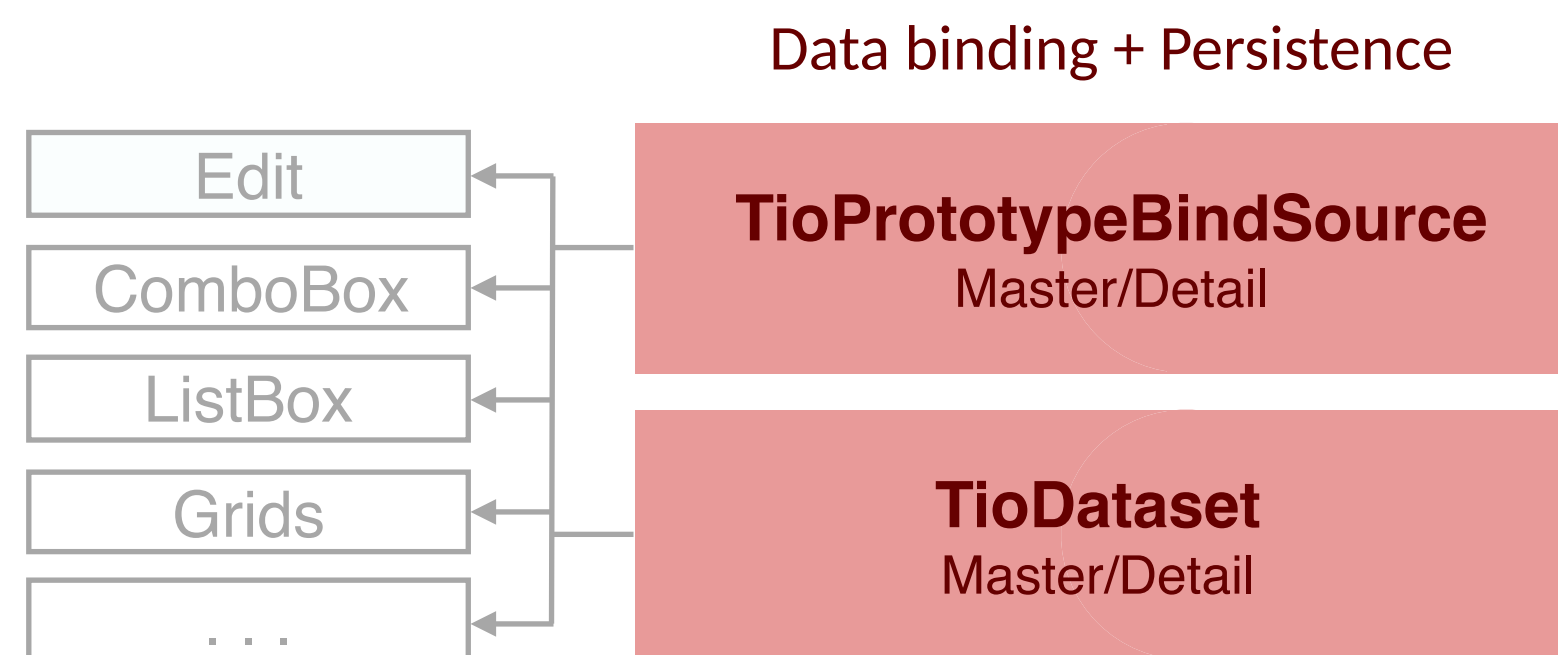


Timeline

o-o-o-o→bycomponents



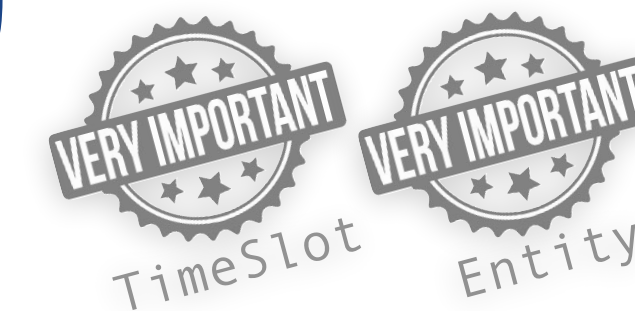
SimpleView



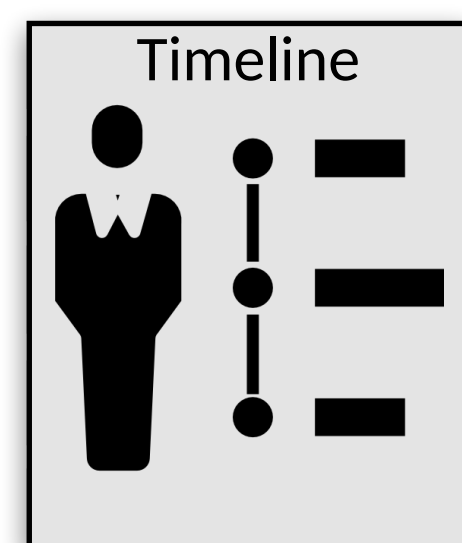


Timeline

bycomponents



SimpleView



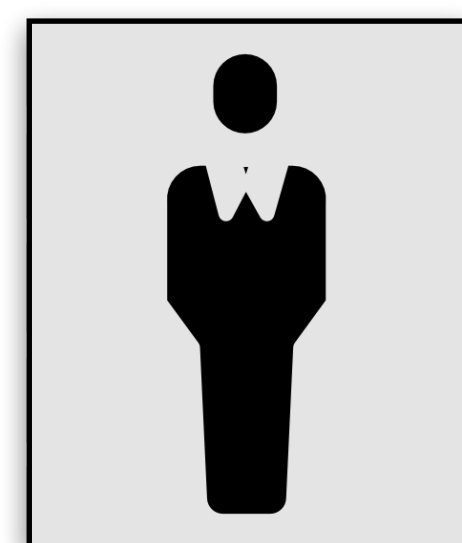
Data binding + Persistence

TioPrototypeBindSource
Master/Detail

TioDataset
Master/Detail

ETMfor (property)

SimpleView



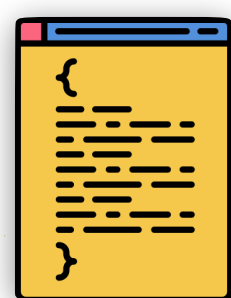
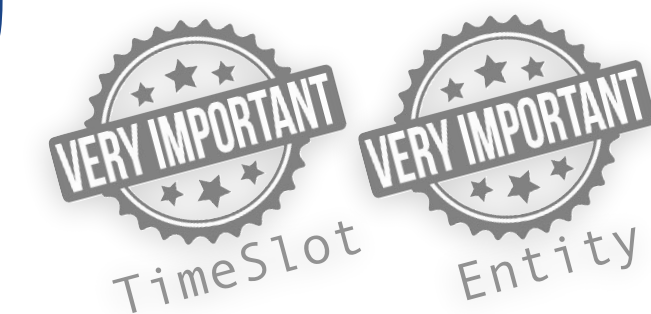
Data binding + Persistence

TioPrototypeBindSource
Master/Detail

TioDataset
Master/Detail



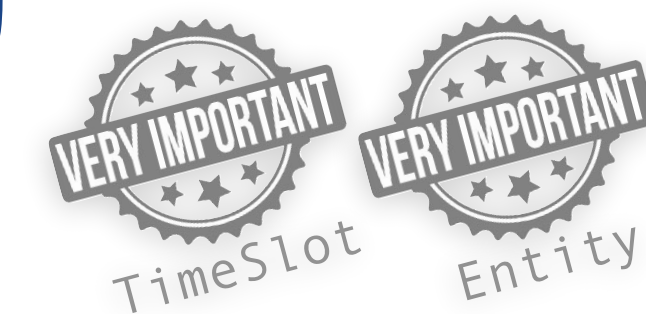
Diff



bycode



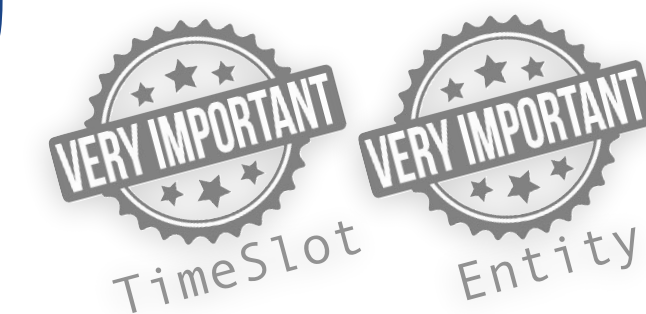
bycomponents



```
var
  LPerson: TPerson;
  LTimeline: TioEtmTimeline;
  LDiff: String;
begin
  ...
  LDiff := io.etm.Diff(LPerson, LTimeline[0]);
  ...
end;
```

```
unit iORM.Attributes;
...
// A Timeline is a list of time slots
TioEtmTimeline = TObjectList<TioEtmCustomTimeSlot>;
...
```

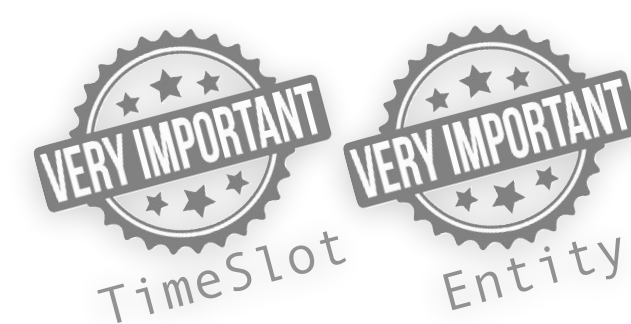
```
{
  "$etm_status": "updated",
  "$etm_class": "TPerson",
  "$etm_id": 1,
  "PhoneNumber": {
    "$etm_old_value": "(555) 555-1234",
    "$etm_new_value": "(888) 888-5678"
  },
  "ObjVersion": {
    "$etm_old_value": 1,
    "$etm_new_value": 2
  }
}
```



```
var
  LPerson: TPerson;
  LTimeline: TioEtmTimeline;
  LDiff: String;
begin
  ...
  LDiff := io.etm.Diff(LPerson, LTimeline[0]);
  ...
end;
```

```
unit iORM.Attributes;
...
// A Timeline is a list of time slots
TioEtmTimeline = TObjectList<TioEtmCustomTimeSlot>;
...
```

```
{
  "$etm_status": "updated",
  "$etm_class": "TPerson",
  "$etm_id": 1,
  "PhoneNumber": {
    "$etm_old_value": "(555) 555-1234",
    "$etm_new_value": "(888) 888-5678"
  },
  "ObjVersion": {
    "$etm_old_value": 1,
    "$etm_new_value": 2
  }
}
```



```
var
  LPerson: TPerson;
  LTimeline: TioEtmTimeline;
  LDiff: String;
begin
  ...
  LDiff := io.etm.Diff(LPerson, LTimeline[0], dmTwoway);
  ...
end;
```

```
unit iORM.Attributes;
...
// A Timeline is a list of time slots
TioEtmTimeline = TObjectList<TioEtmCustomTimeSlot>;
...
```

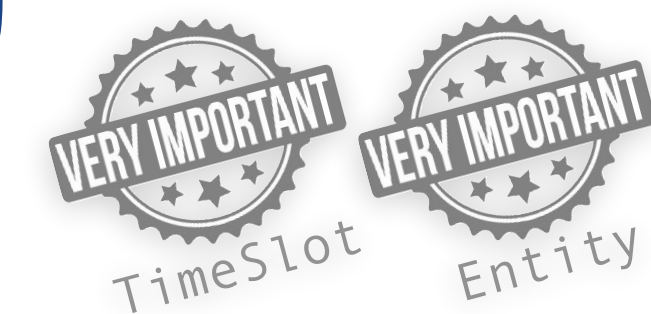
```
unit iORM.ETM.Interfaces;
...
//Diff mode
TioEtmDiffMode = (dmOneway, dmTwoway);
...
```

```
{
  "$etm_status": "updated",
  "$etm_class": "TPerson",
  "$etm_id": 1,
  "PhoneNumber": {
    "$etm_old_value": "(555) 555-1234",
    "$etm_new_value": "(888) 888-5678"
  },
  "ObjVersion": {
    "$etm_old_value": 1,
    "$etm_new_value": 2
  }
}
```

```
{
  "$etm_old_value": {
    "$etm_status": "updated",
    "$etm_class": "TCustomer",
    "$etm_id": 1,
    "PhoneNumber": "(555) 555-1234",
    "ObjVersion": 1
  },
  "$etm_new_value": {
    "$etm_status": "updated",
    "$etm_class": "TCustomer",
    "$etm_id": 1,
    "PhoneNumber": "(888) 888-5678",
    "ObjVersion": 2
  }
}
```




Diff bycodeoverloads



```
unit iORM.ETM.Engine;  
...  
TioEtmEngine = class  
public
```

```
...
```

```
// Diff
```

```
Diff(ANewestObj:TObject; AOldestTimeSlot:TioEtmCustomTimeSlot; DiffMode:TioEtmDiffMode=dmOneway; MoreInfo:Boolean=False):String;
```

```
Diff(ANewestIntf:IInterface; AOldestTimeSlot:TioEtmCustomTimeSlot; ...):String; overload;
```

```
Diff(ANewestTimeSlot, AOldestTimeSlot: TioEtmCustomTimeSlot; ...):String; overload;
```

```
// DiffAsJsonObject
```

```
DiffAsJsonObject(...): TJSONObject; overload; // 3 overloads
```

```
// DiffToFile
```

```
DiffToFile(AFileName: String; ...); overload; // 3 overloads
```

```
// DiffToStream
```

```
DiffToStream(ATargetStream: TStream; ...); overload; // 3 overloads
```

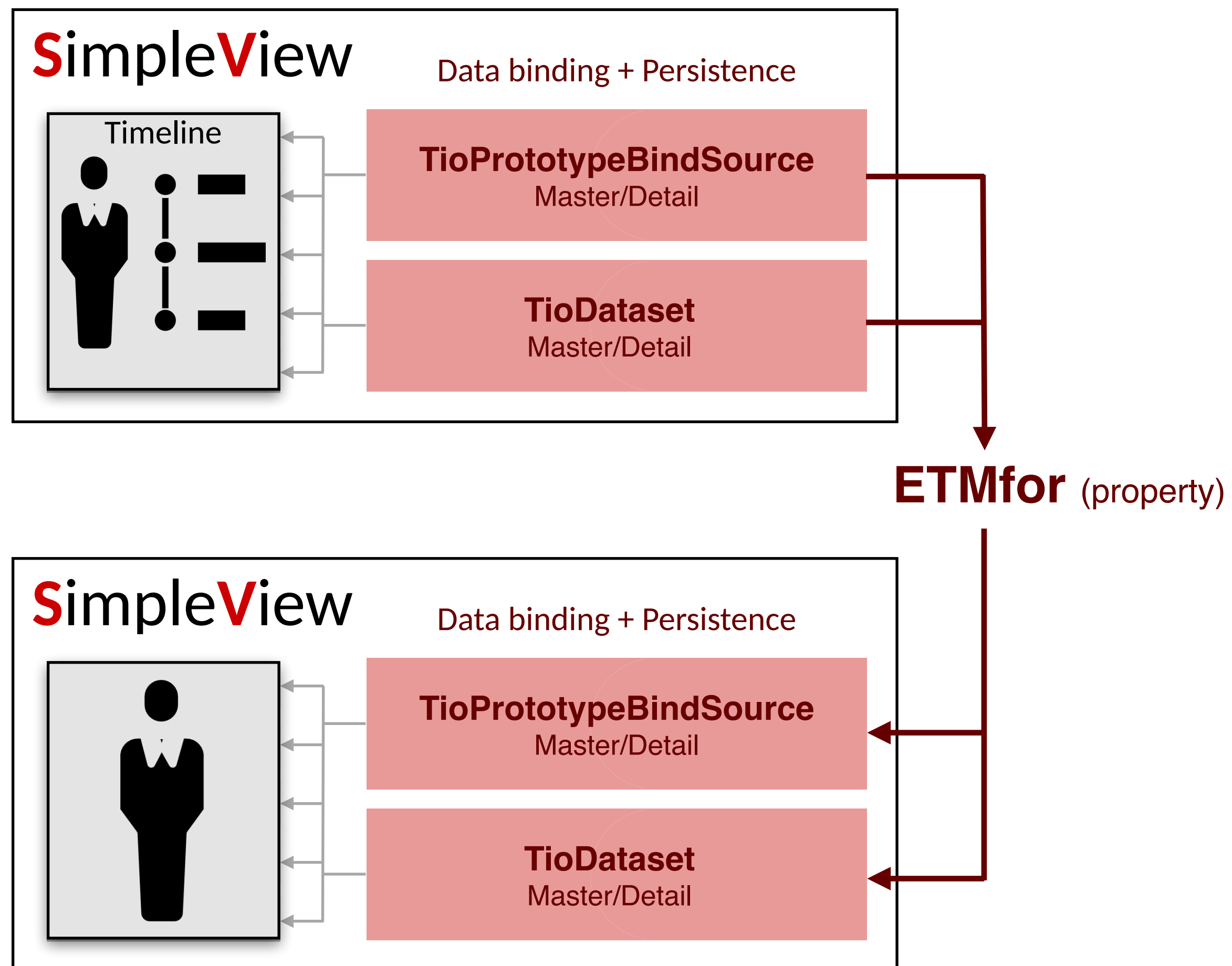
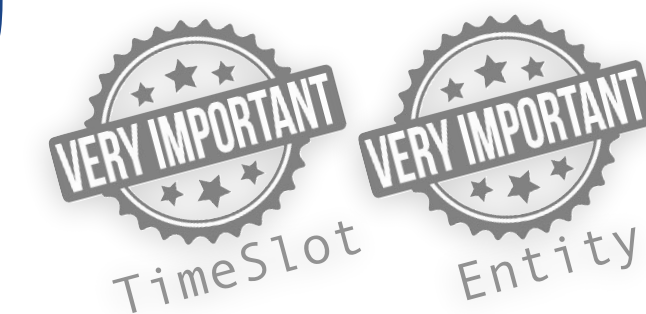
```
...
```

```
end;
```

```
unit iORM.ETM.Interfaces;  
...  
//Diff mode  
TioEtmDiffMode = (dmOneway, dmTwoway);  
...
```



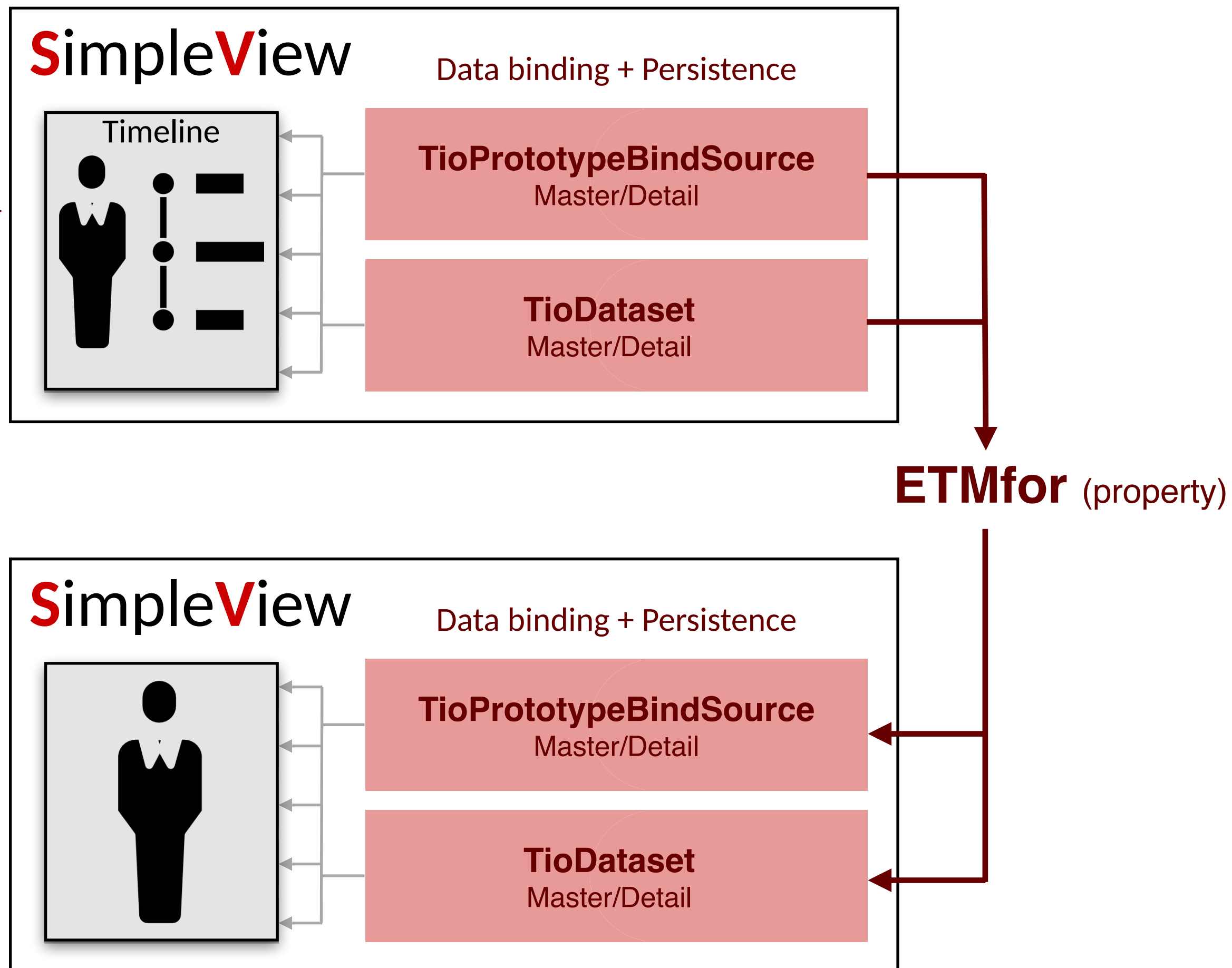
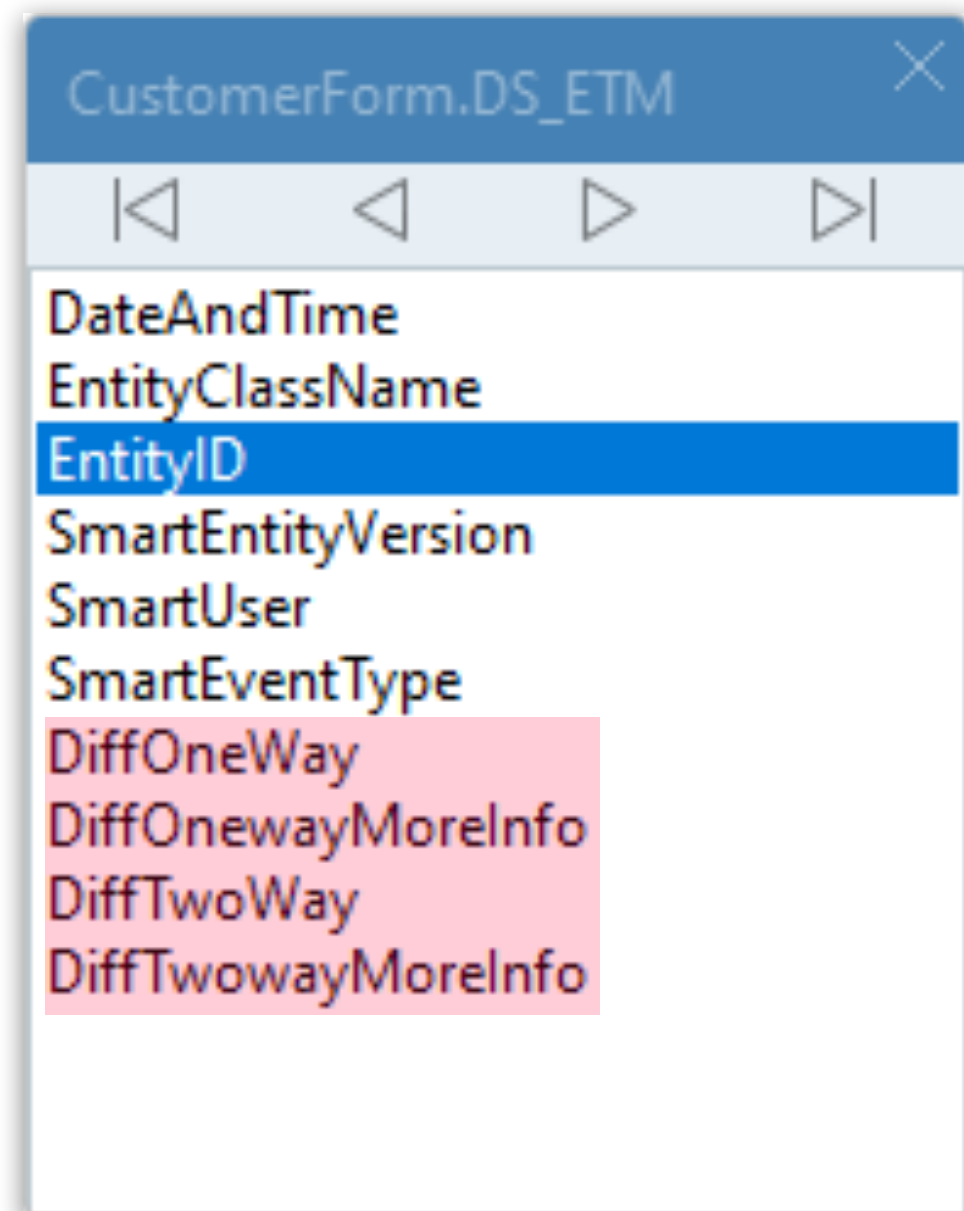
Diff bycomponents



```
unit iORM.Attributes;  
  
TioEtmCustomTimeSlot = class  
...  
public  
    property ID;  
    property DateAndTime;  
    property EventType;  
    property EntityClassName;  
    property EntityID;  
    property EntityVersion;  
    property RevertedFromVersion;  
    property EntityState;  
    property RemoteEntityState;  
    property ConflictType;  
    property Username;  
    property UserID;  
...  
end;
```



Diff bycomponents



```
unit iORM.Attributes;

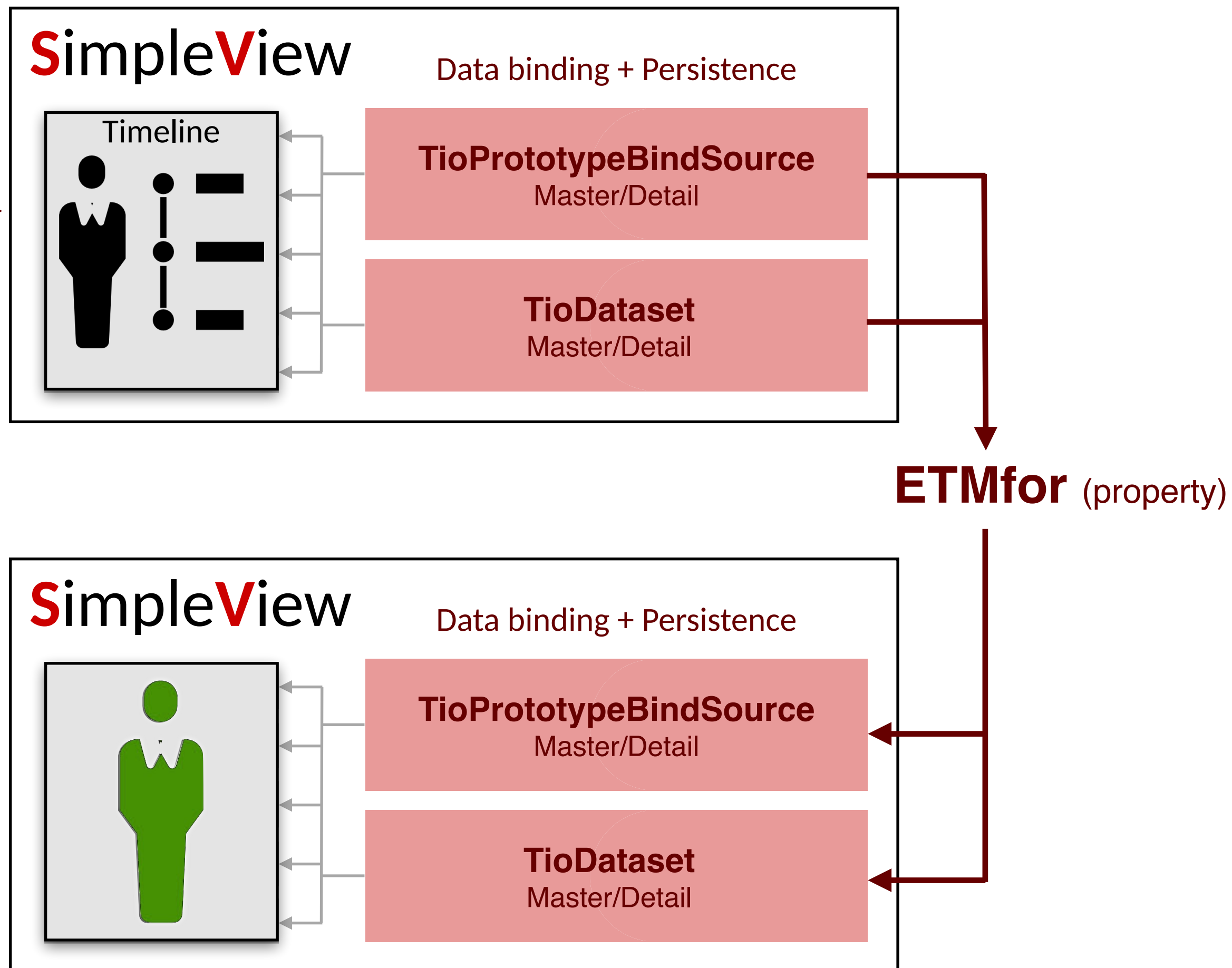
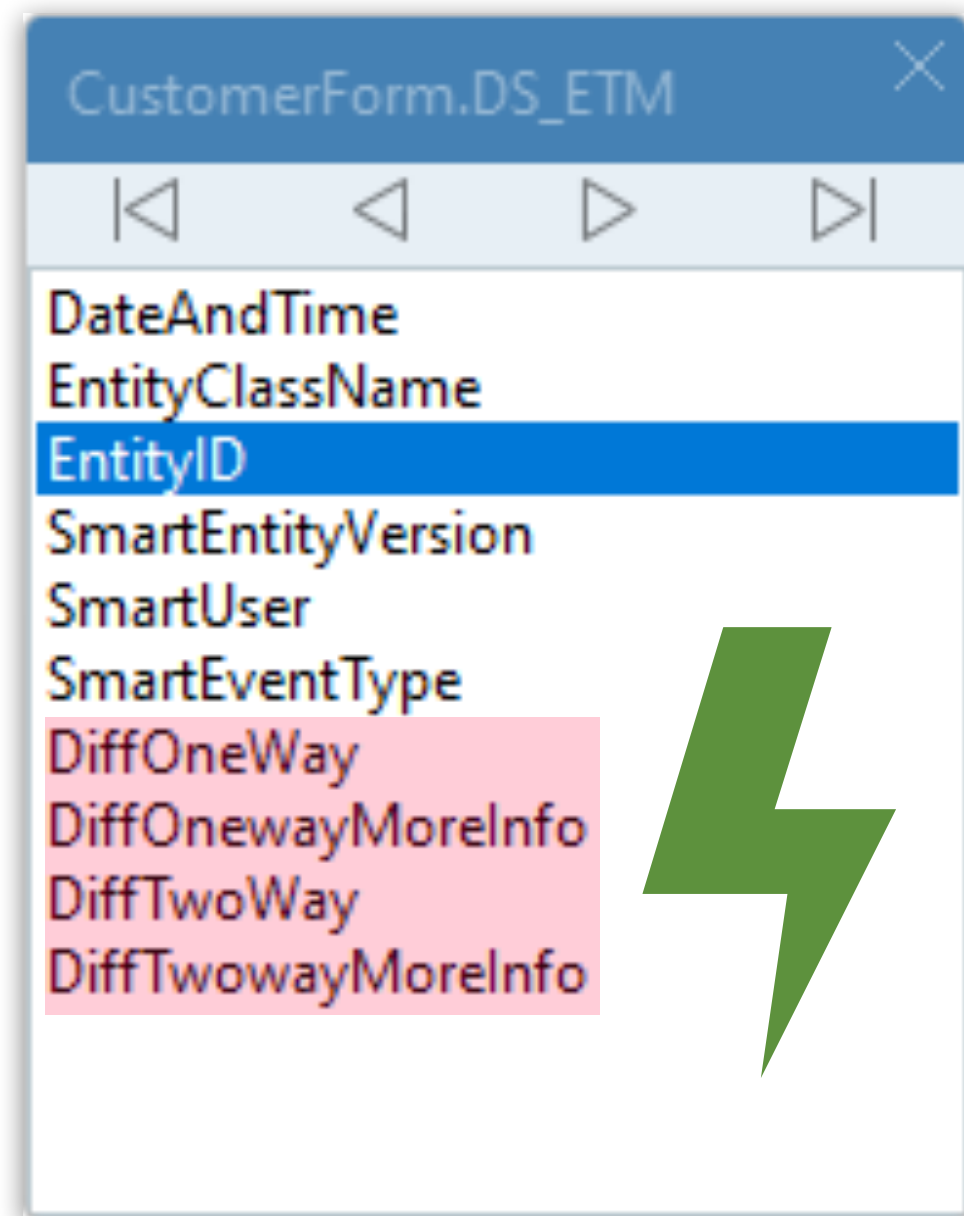
TioEtmCustomTimeSlot = class
...
public
    property ID;
    property DateAndTime;
    property EventType;
    property EntityClassName;
    property EntityID;
    property EntityVersion;
    property RevertedFromVersion;
    property EntityState;
    property RemoteEntityState;
    property ConflictType;
    property Username;
    property UserID;
...

// Diff properties
property DiffOneway;
property DiffOnewayMoreInfo;
property DiffTwoWay;
property DiffTwowayMoreInfo;
...
```

etmRepository = ioEntity;



Diff bycomponents

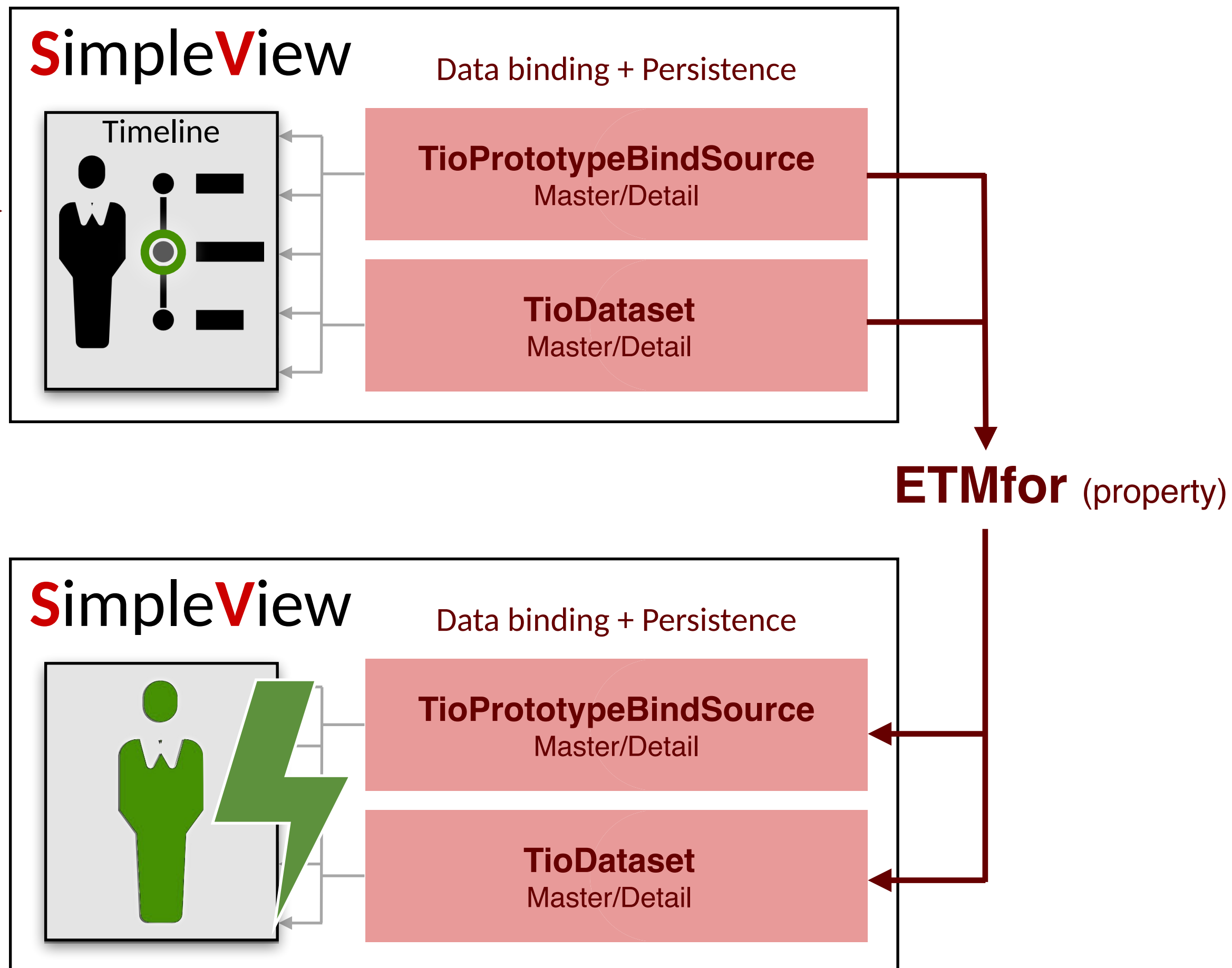
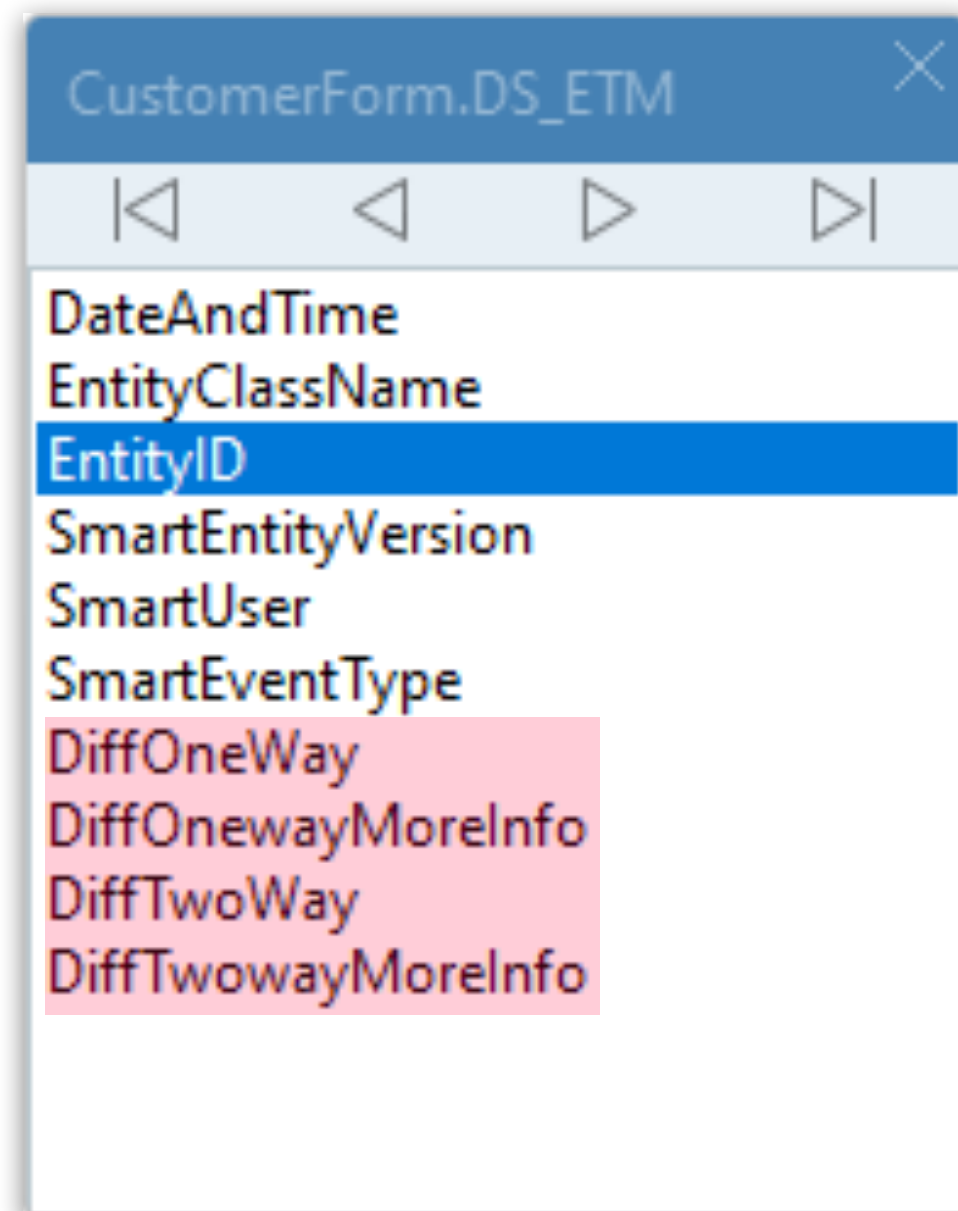


```
unit iORM.Attributes;  
  
TioEtmCustomTimeSlot = class  
...  
public  
    property ID;  
    property DateAndTime;  
    property EventType;  
    property EntityClassName;  
    property EntityID;  
    property EntityVersion;  
    property RevertedFromVersion;  
    property EntityState;  
    property RemoteEntityState;  
    property ConflictType;  
    property Username;  
    property UserID;  
...  
  
    // Diff properties  
    property DiffOneway;  
    property DiffOnewayMoreInfo;  
    property DiffTwoway;  
    property DiffTwowayMoreInfo;  
...
```

etmRepository = ioEntity;



Diff bycomponents

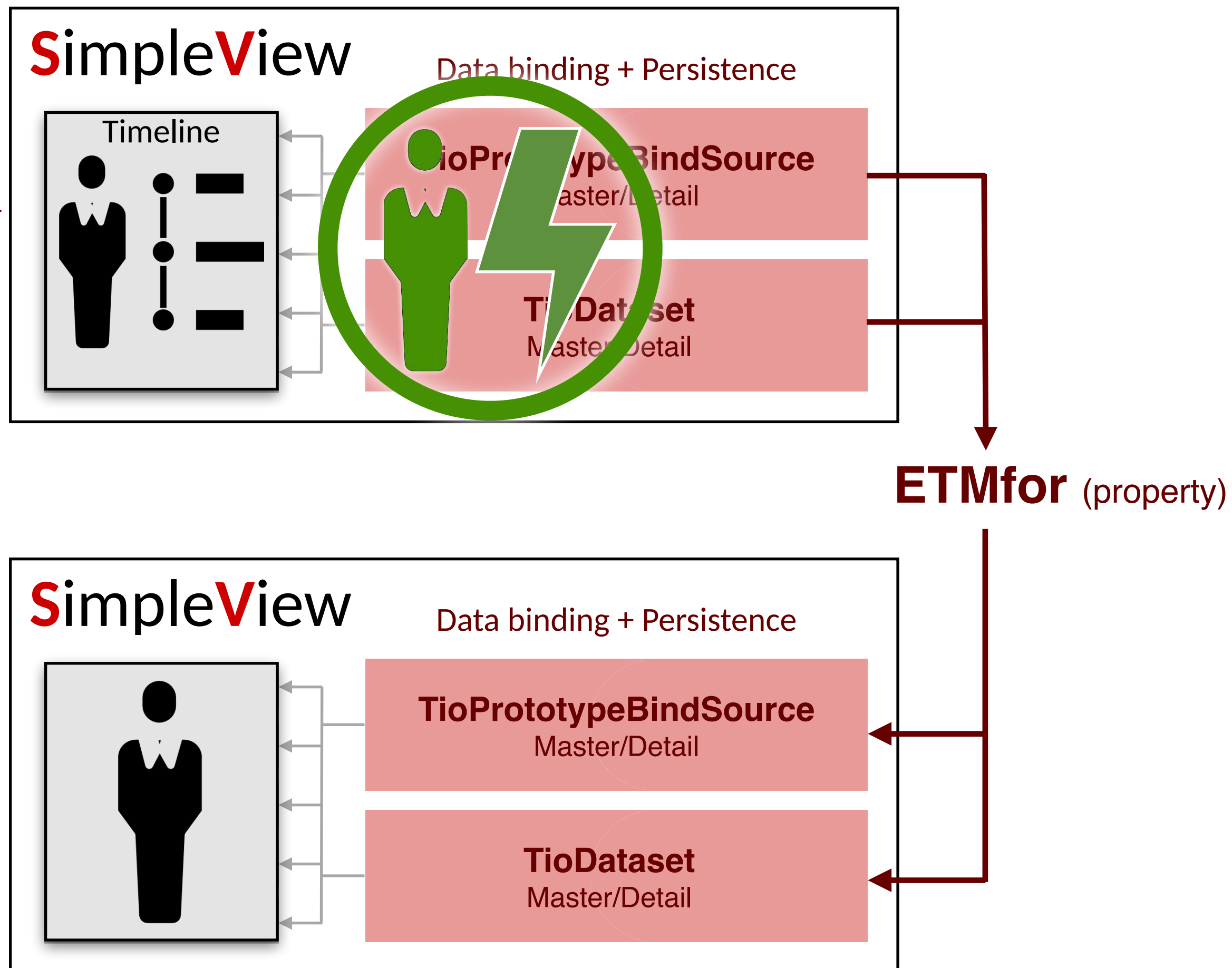
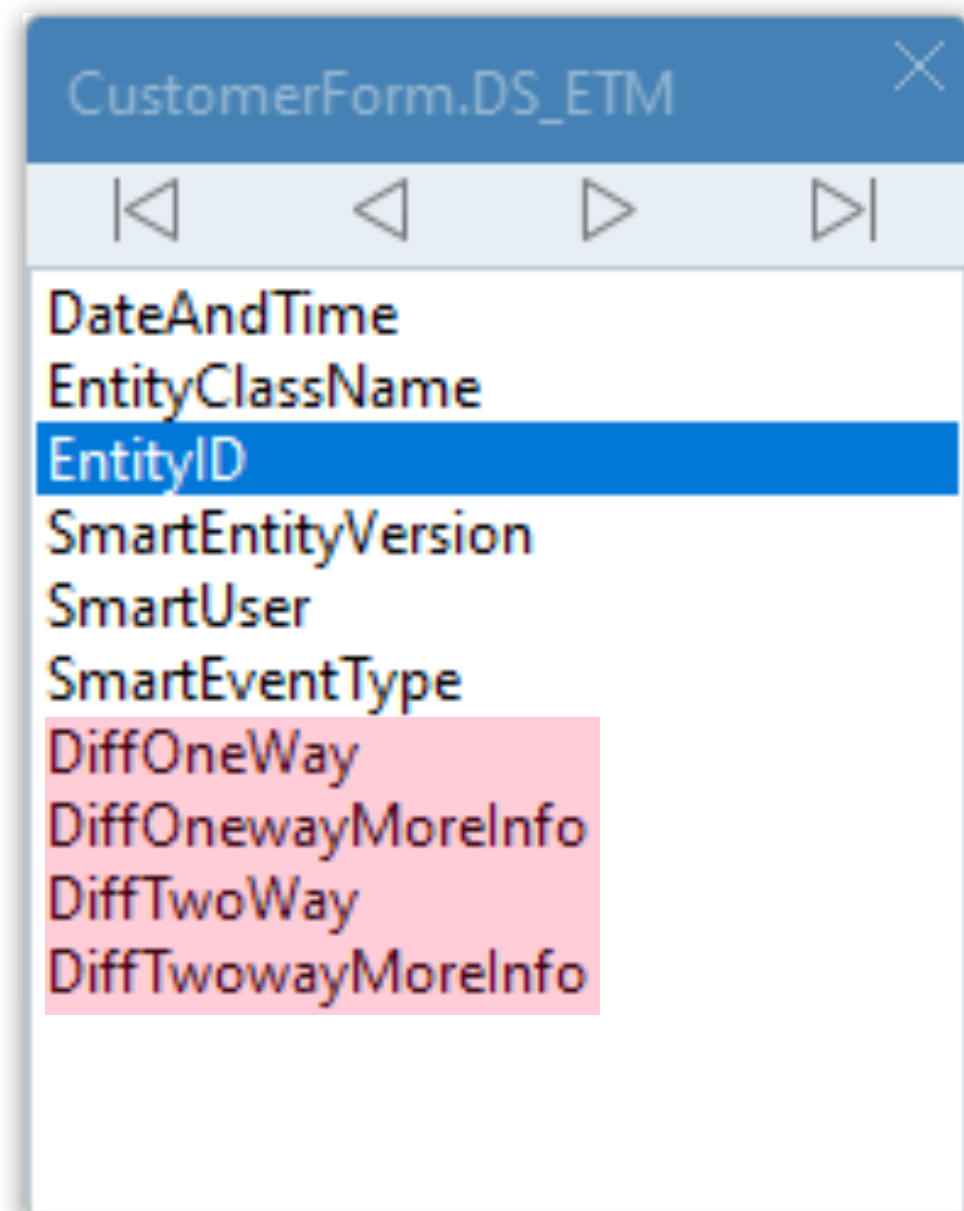


```
unit iORM.Attributes;  
  
TioEtmCustomTimeSlot = class  
...  
public  
    property ID;  
    property DateAndTime;  
    property EventType;  
    property EntityClassName;  
    property EntityID;  
    property EntityVersion;  
    property RevertedFromVersion;  
    property EntityState;  
    property RemoteEntityState;  
    property ConflictType;  
    property Username;  
    property UserID;  
...  
  
    // Diff properties  
    property DiffOneway;  
    property DiffOnewayMoreInfo;  
    property DiffTwoway;  
    property DiffTwowayMoreInfo;  
...
```

etmRepository = ioEntity;



Diff bycomponents

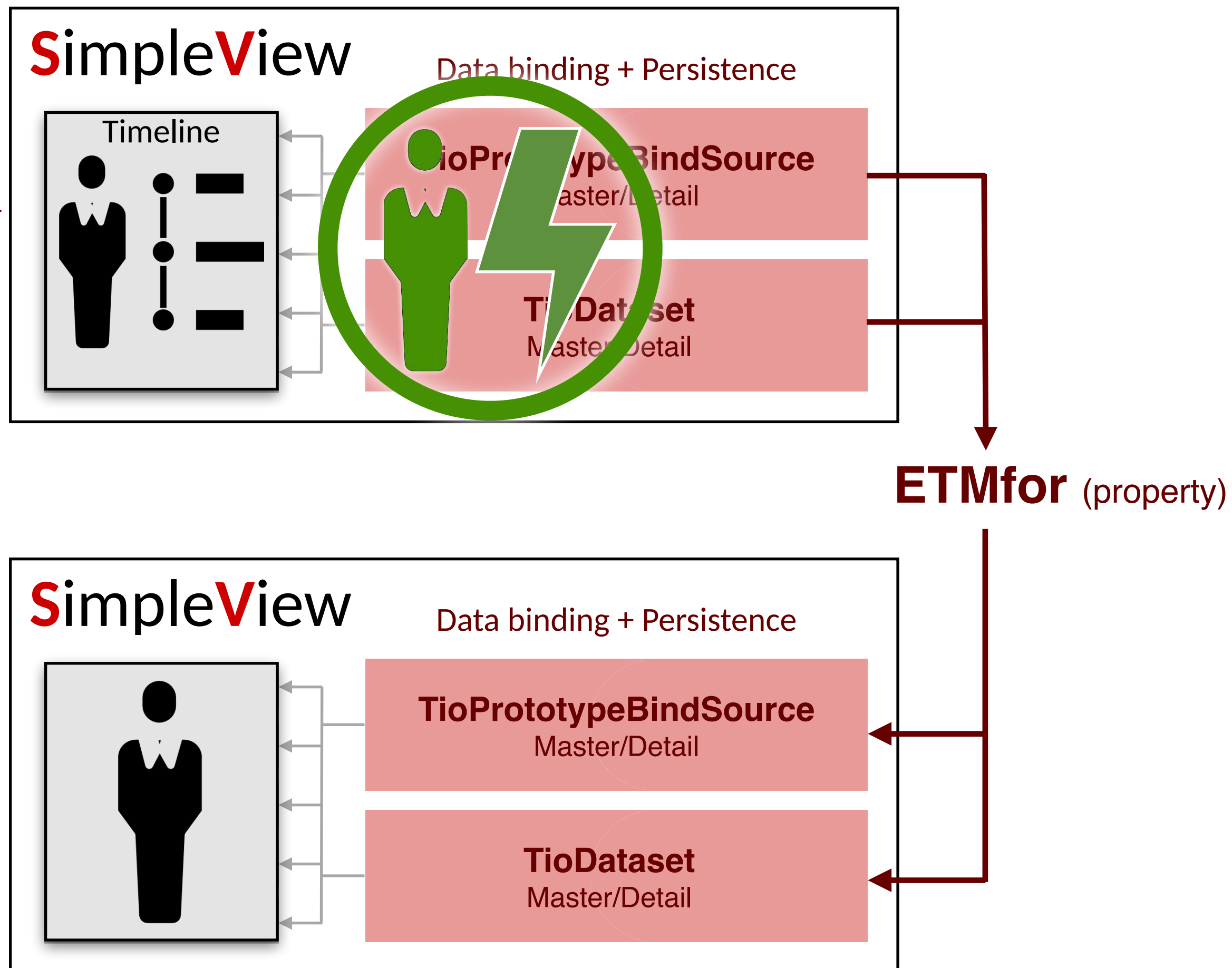
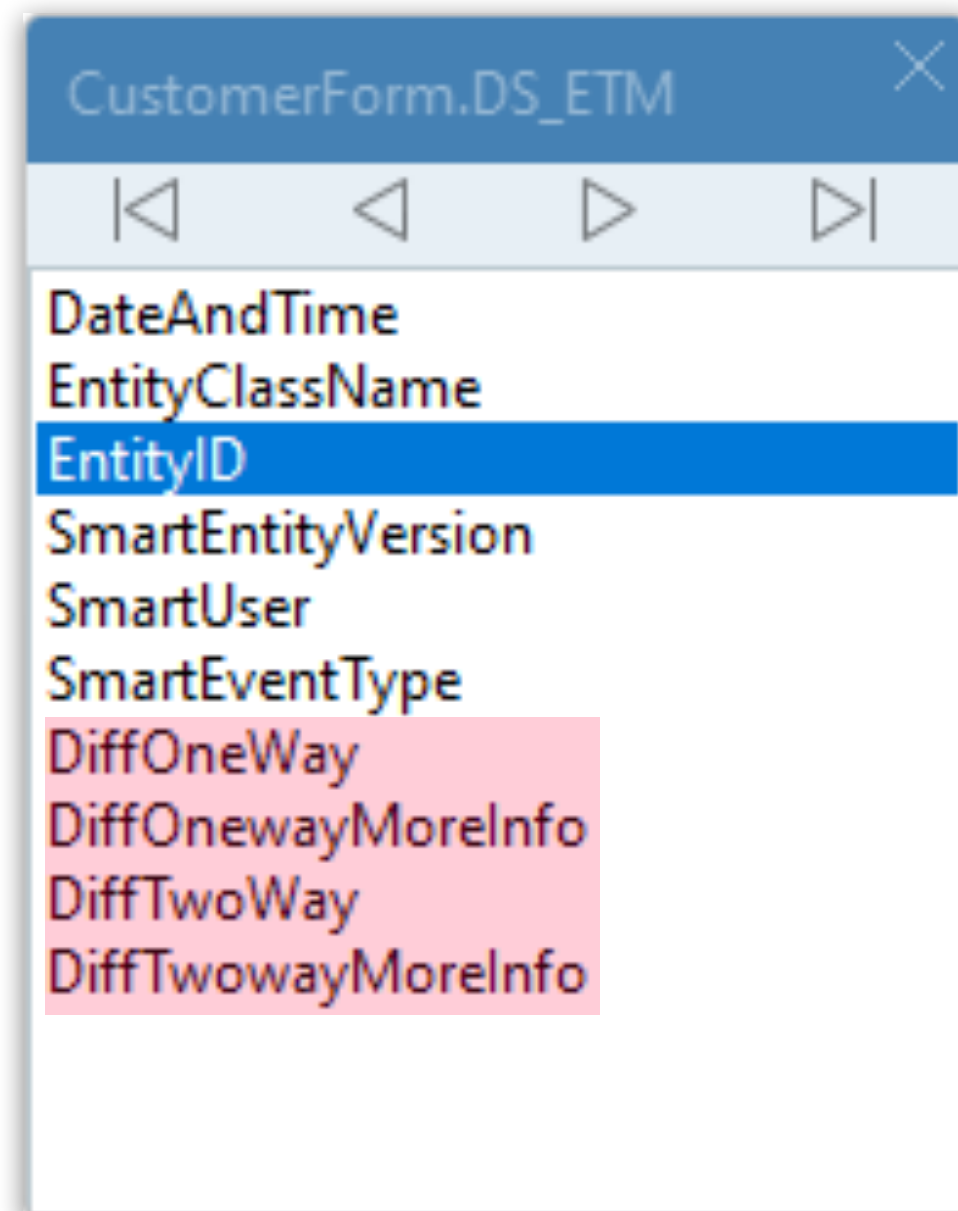


```
unit iORM.Attributes;  
  
TioEtmCustomTimeSlot = class  
...  
public  
    property ID;  
    property DateAndTime;  
    property EventType;  
    property EntityClassName;  
    property EntityID;  
    property EntityVersion;  
    property RevertedFromVersion;  
    property EntityState;  
    property RemoteEntityState;  
    property ConflictType;  
    property Username;  
    property UserID;  
...  
  
    // Diff properties  
    property DiffOneway;  
    property DiffOnewayMoreInfo;  
    property DiffTwoway;  
    property DiffTwowayMoreInfo;  
...
```

etmRepository = ioEntity;



Diff bycomponents



```
unit iORM.Attributes;

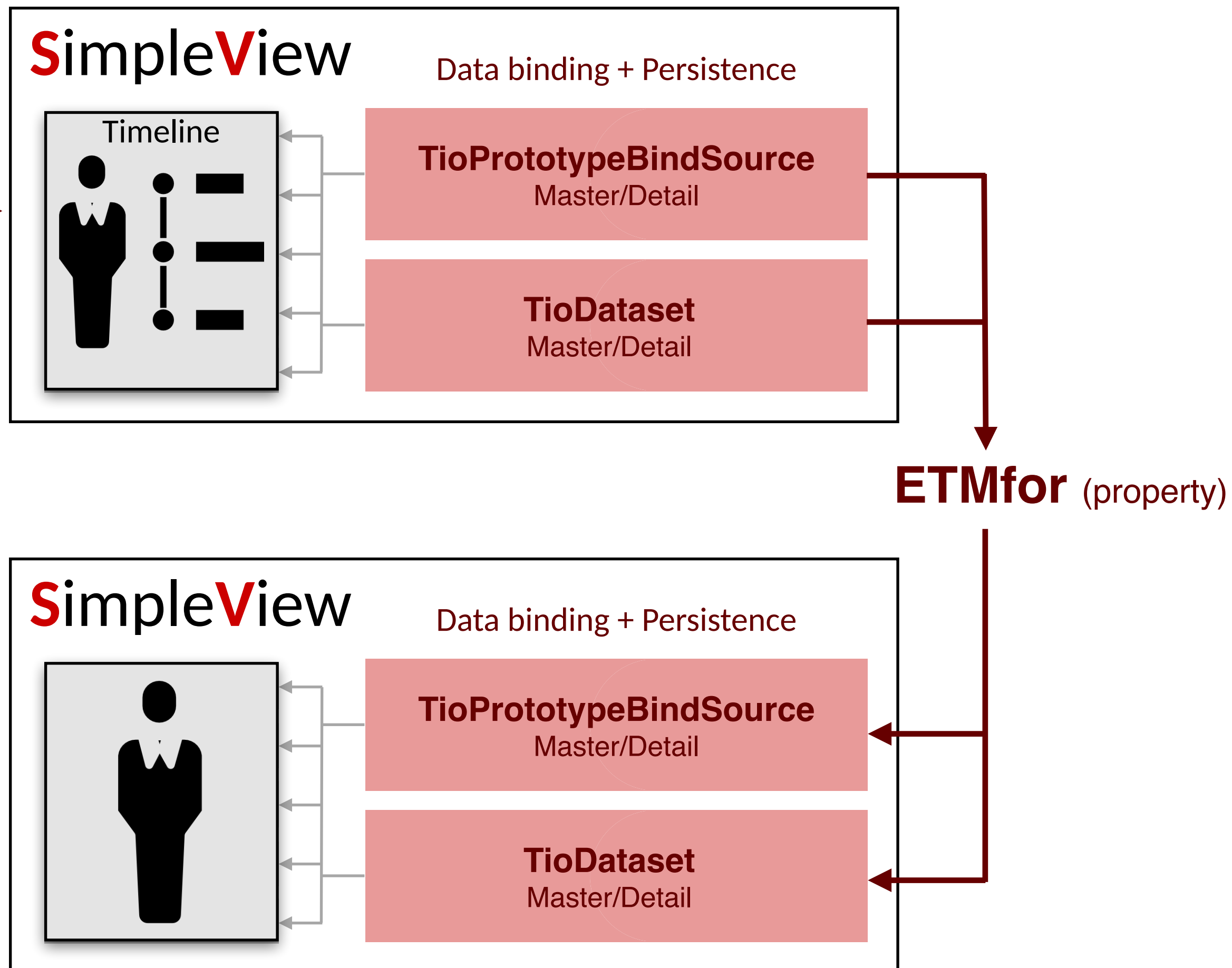
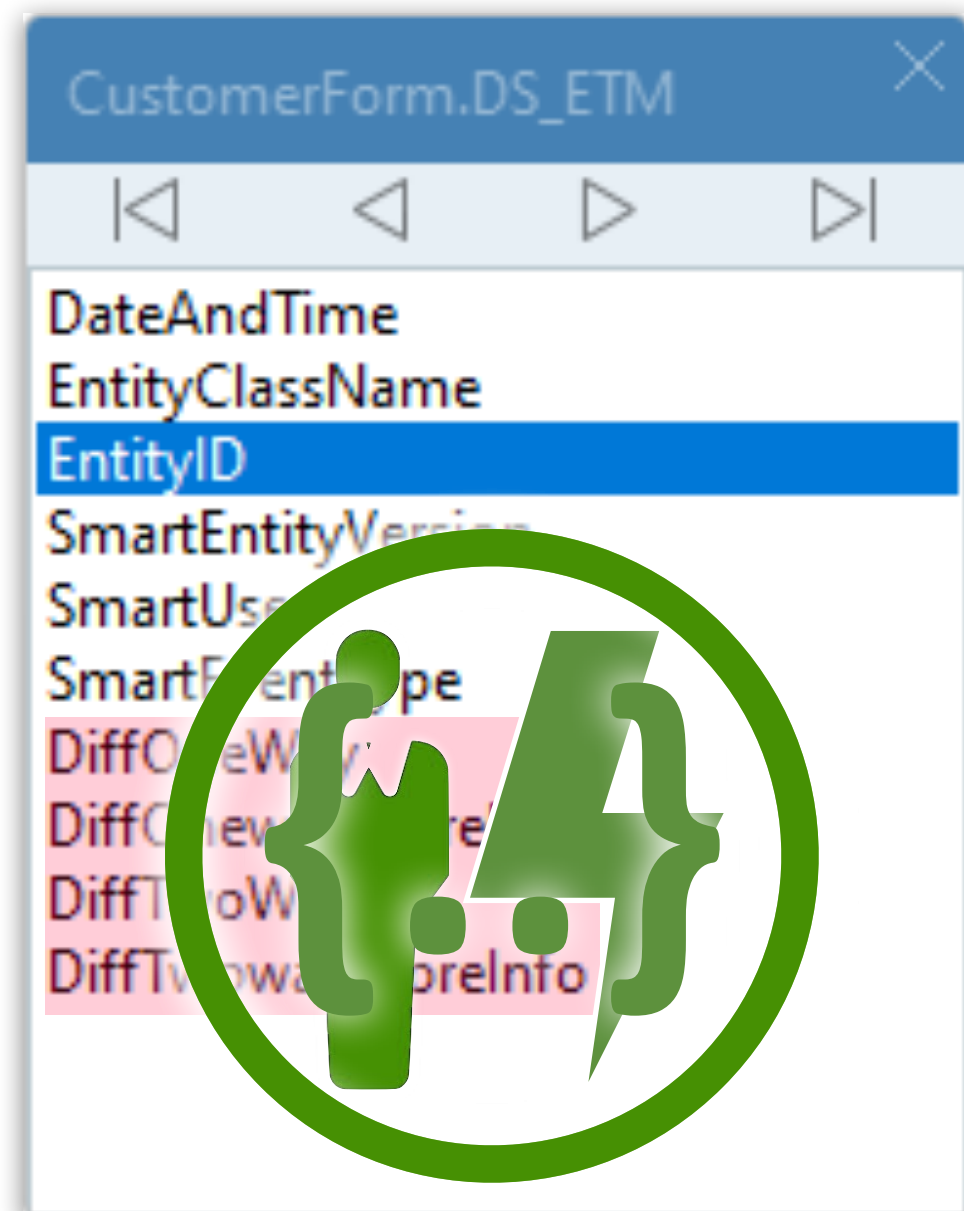
TioEtmCustomTimeSlot = class
...
public
    property ID;
    property DateAndTime;
    property EventType;
    property EntityClassName;
    property EntityID;
    property EntityVersion;
    property RevertedFromVersion;
    property EntityState;
    property RemoteEntityState;
    property ConflictType;
    property Username;
    property UserID;
...

// Diff properties
property DiffOneway;
property DiffOnewayMoreInfo;
property DiffTwoway;
property DiffTwowayMoreInfo;
...
```

etmRepository = ioEntity;



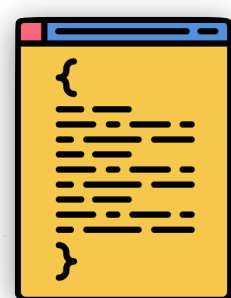
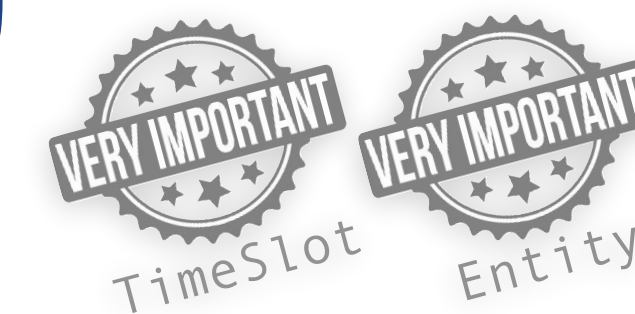
Diff bycomponents



```
unit iORM.Attributes;  
  
TioEtmCustomTimeSlot = class  
...  
public  
    property ID;  
    property DateAndTime;  
    property EventType;  
    property EntityClassName;  
    property EntityID;  
    property EntityVersion;  
    property RevertedFromVersion;  
    property EntityState;  
    property RemoteEntityState;  
    property ConflictType;  
    property Username;  
    property UserID;  
...  
  
    // Diff properties  
    property DiffOneway;  
    property DiffOnewayMoreInfo;  
    property DiffTwoway;  
    property DiffTwowayMoreInfo;  
...
```

etmRepository = ioEntity;

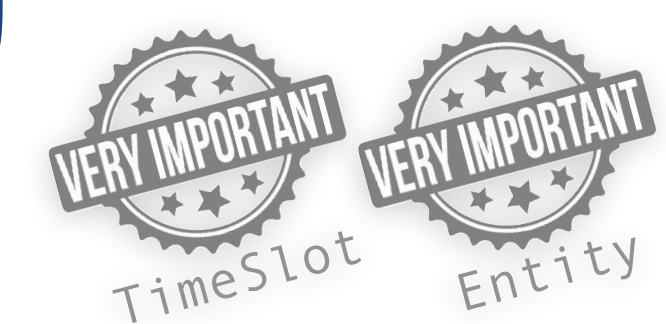




bycode

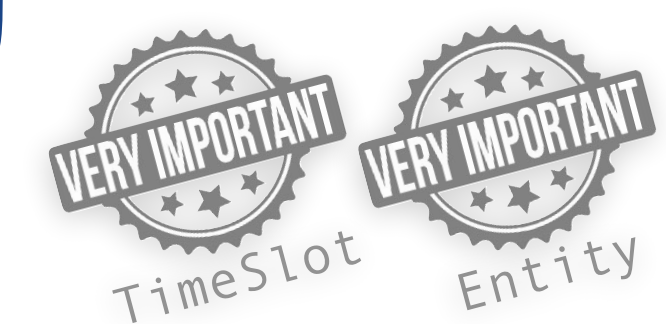


bycomponents



```
var
  LPerson: TPerson;
  LTimeline: TioEtmTimeline;
begin
  ...
  LPerson := io.etm.RevertObject<TPerson>(LTimeline[0]);
  ...
end;
```

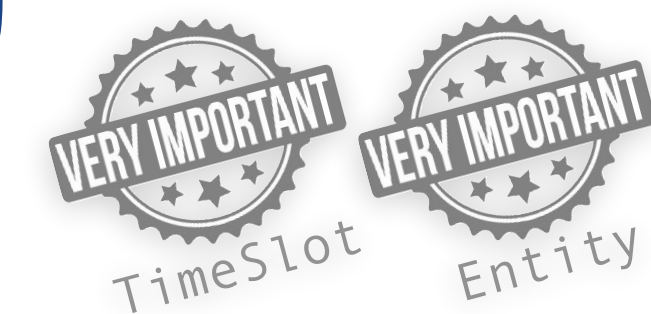
```
unit iORM.Attributes;
...
// A TimeLine is a list of time slots
TioEtmTimeline = TObjectList<TioEtmCustomTimeSlot>;
...
```



```
var
  LPerson: TPerson;
  LTimeline: TioEtmTimeline;
begin
  ...
  LPerson := io.etm.RevertObject<TPerson>(LTimeline[0]);
  ...
end;
```

```
unit iORM.Attributes;
...
// A Timeline is a list of time slots
TioEtmTimeline = TObjectList<TioEtmCustomTimeSlot>;
...
```





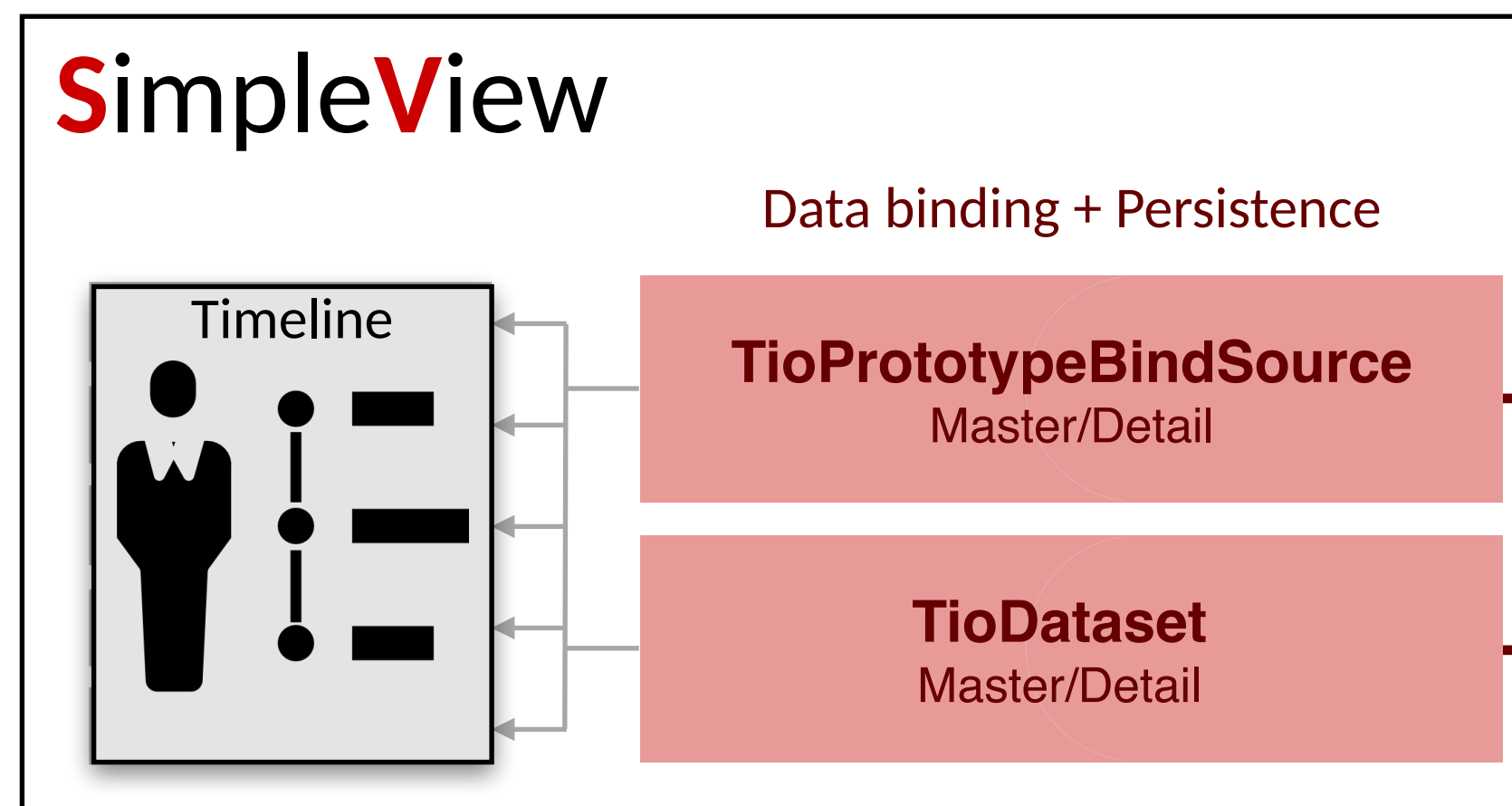
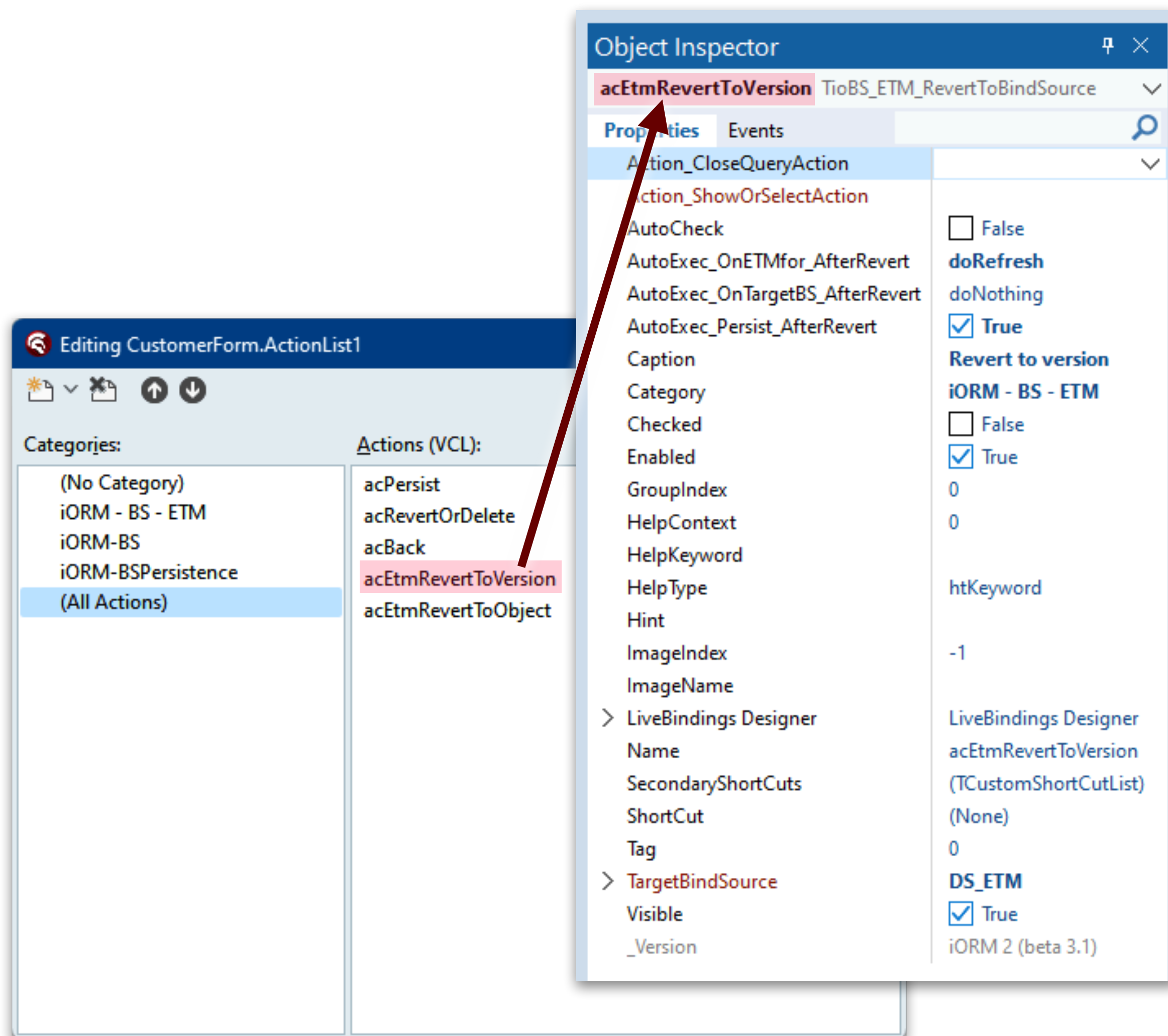
```
var
  LPerson: TPerson;
  LTimeline: TioEtmTimeline;
begin
  ...
  LPerson := io.etm.RevertObject<TPerson>(LTimeline[0], True);
  ...
end;
```

```
unit iORM.Attributes;
...
// A TimeLine is a list of time slots
TioEtmTimeline = TObjectList<TioEtmCustomTimeSlot>;
...
```

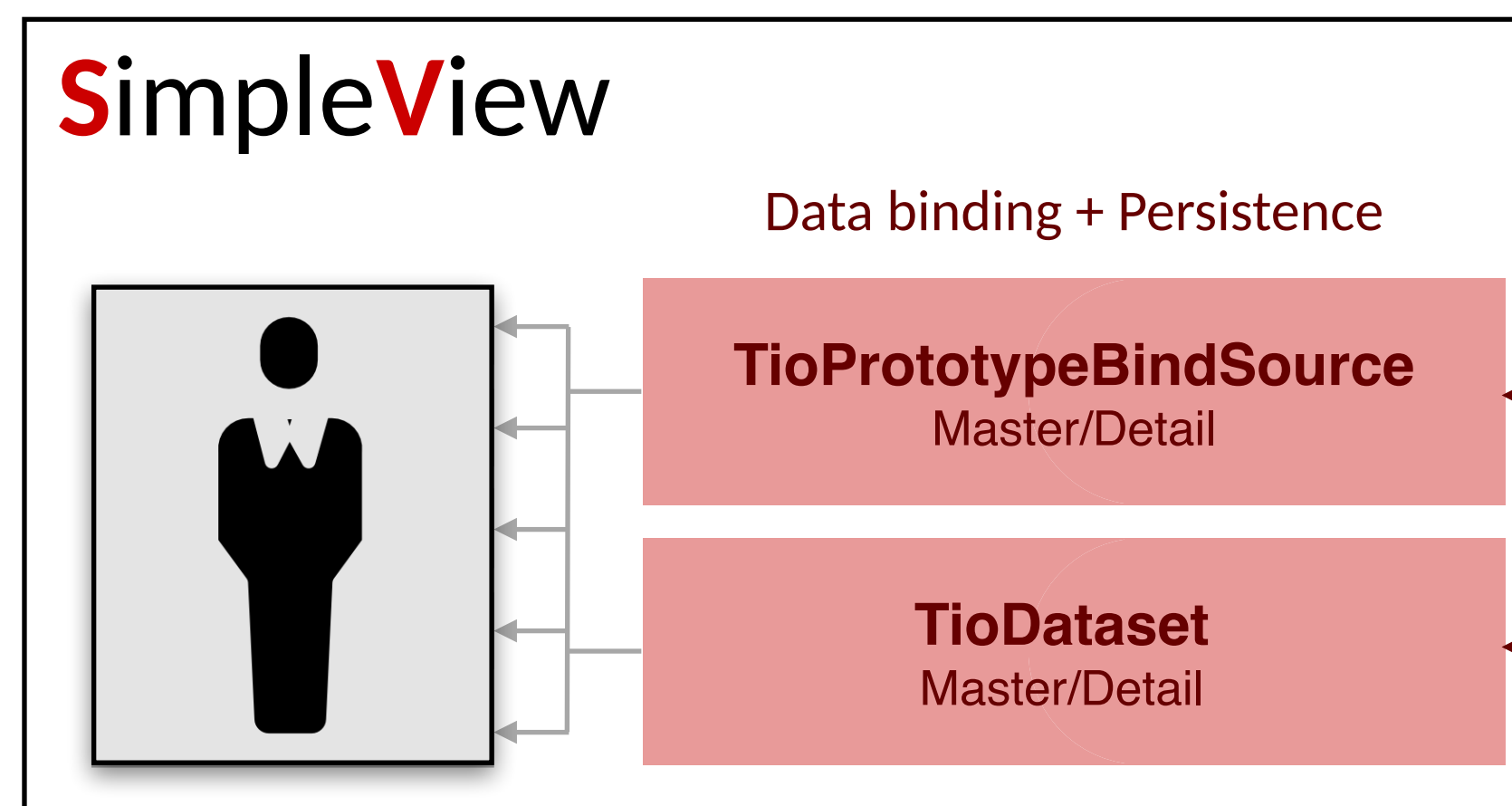
APersistImmediately: Boolean = False



```
unit iORM.ETM.Engine;  
...  
TioEtmEngine = class  
public  
    ...  
    // Revert  
    RevertObject<T: class>(ATimeSlot: TioEtmCustomTimeSlot; APersistImmediately: Boolean = False): T; overload;  
    RevertObject(ATimeSlot: TioEtmCustomTimeSlot; APersistImmediately: Boolean = False): TObject; overload;  
  
    // RevertToObject  
    RevertToObject(ATargetObj: TObject; ATimeSlot: TioEtmCustomTimeSlot; APersistImmediately: Boolean = False); overload;  
    RevertToObject(ATargetIntf: IInterface; ATimeSlot: TioEtmCustomTimeSlot; APersistImmediately: Boolean = False); overload;  
  
    // RevertToDB  
    RevertToDB(ATimeSlot: TioEtmCustomTimeSlot);  
  
    // RevertToBindSource  
    RevertToBindSource(ATimeSlot: TioEtmCustomTimeSlot; ATargetBindSource: IioMasterBindSource; APersistImmediately: Boolean=False);  
    ...  
end;
```



ETMfor (property)





Object Inspector

acEtmRevertToVersion TioBS_ETM_RevertToBindSource

Properties Events

Action_CloseQueryAction

Action_ShowOrSelectAction

AutoCheck ☐ False

AutoExec_OnETMfor_AfterRevert doRefresh

AutoExec_OnTargetBS_AfterRevert doNothing

AutoExec_Persist_AfterRevert ☒ True

Caption Revert to version

Category iORM - BS - ETM

Checked ☐ False

Enabled ☒ True

GroupIndex 0

HelpContext

HelpKeyword

HelpType

Hint

ImageIndex -1

ImageName

> LiveBindings Designer

Name

SecondaryShortCuts

ShortCut

Tag

> TargetBindSource DS_ETM

Visible ☒ True

_Version iORM 2 (beta 3.1)

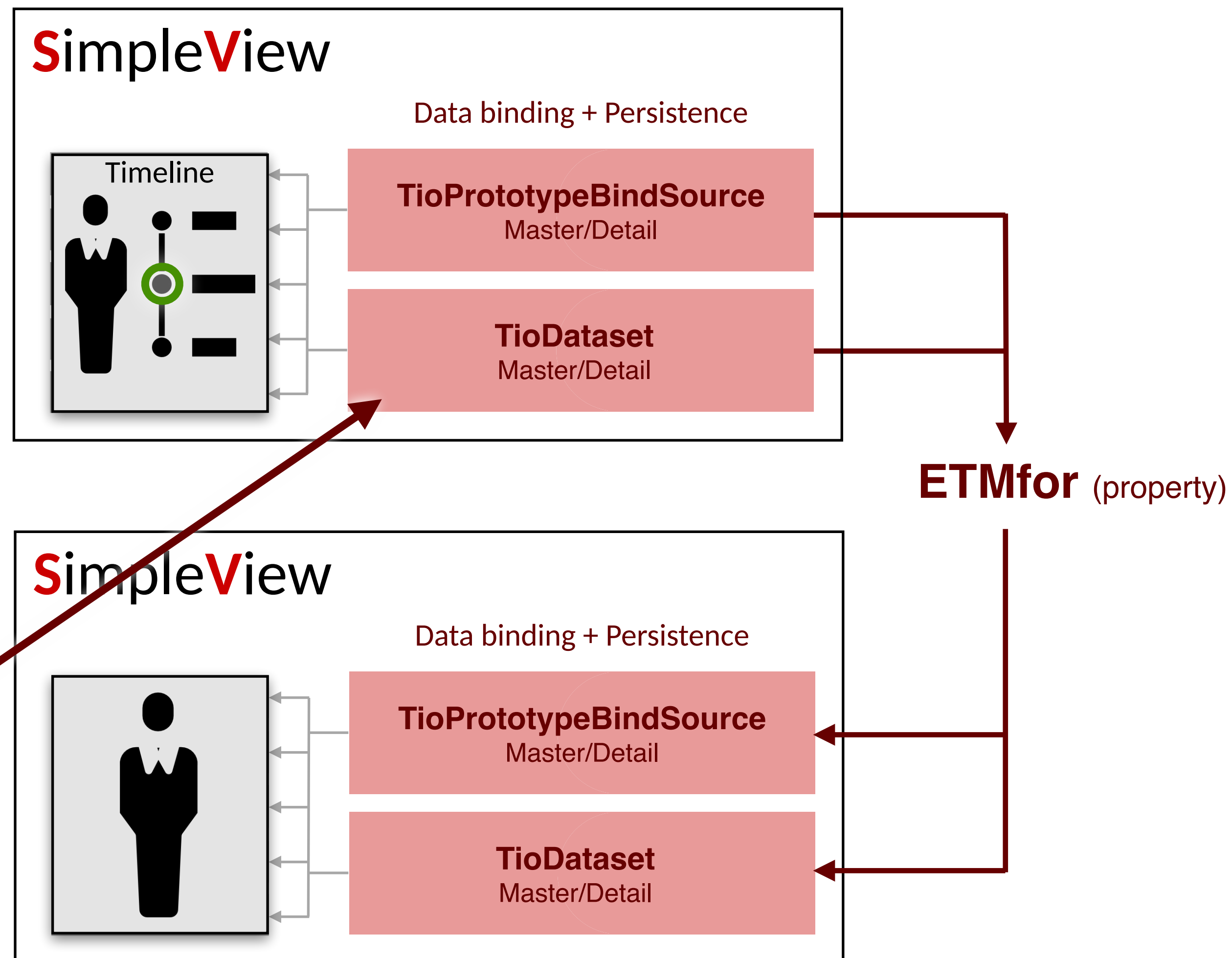
Editing CustomerForm.ActionList1

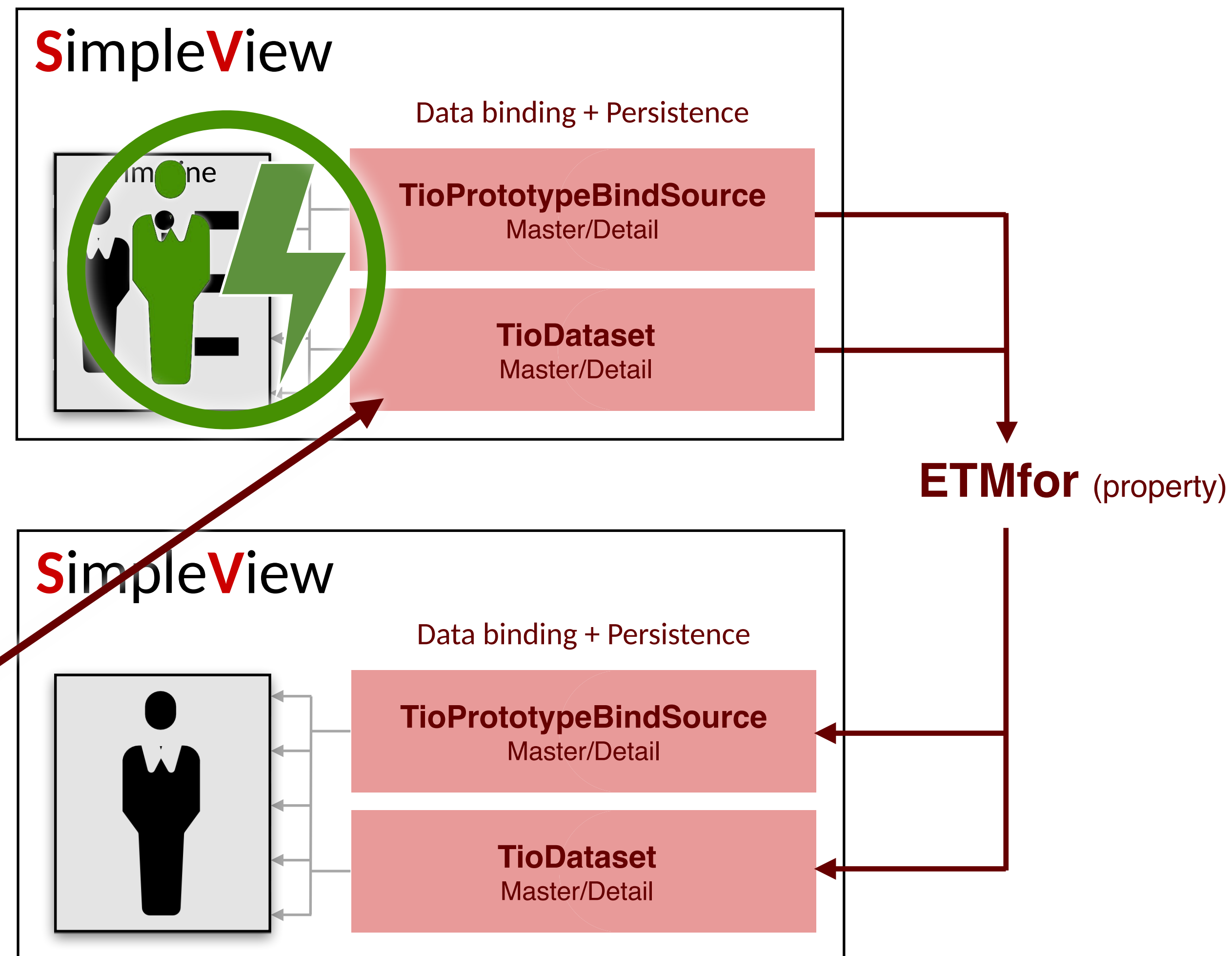
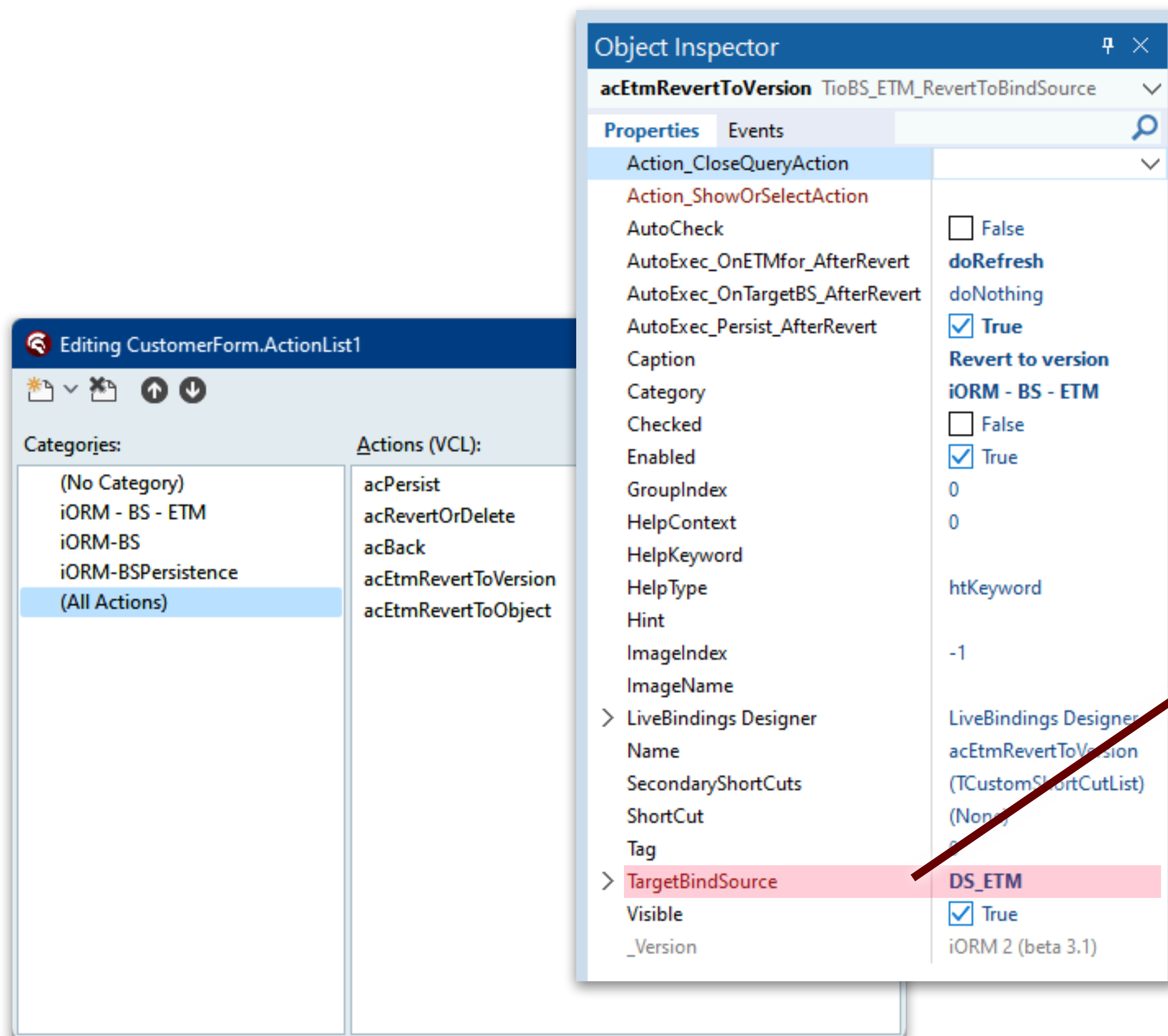
Categories:

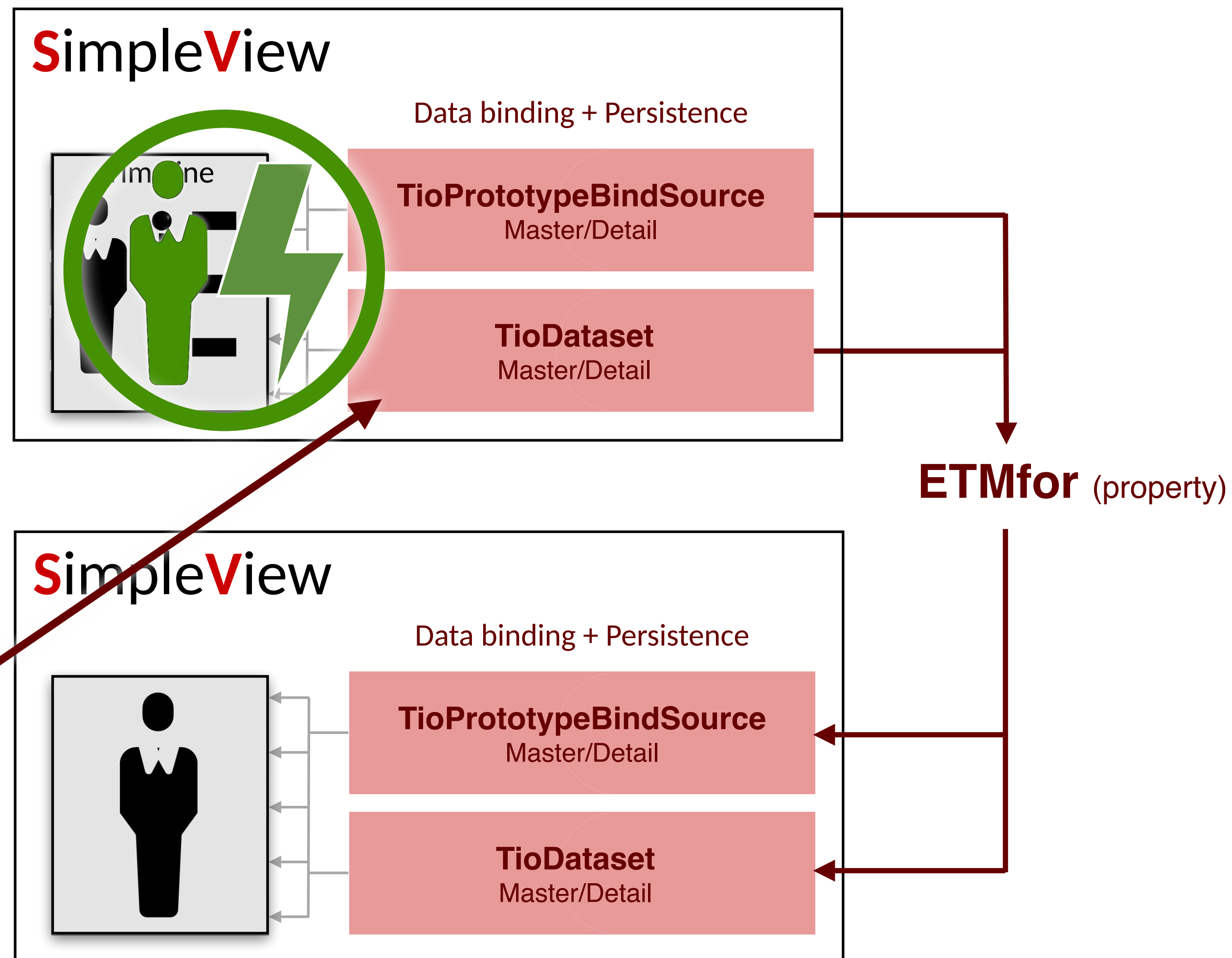
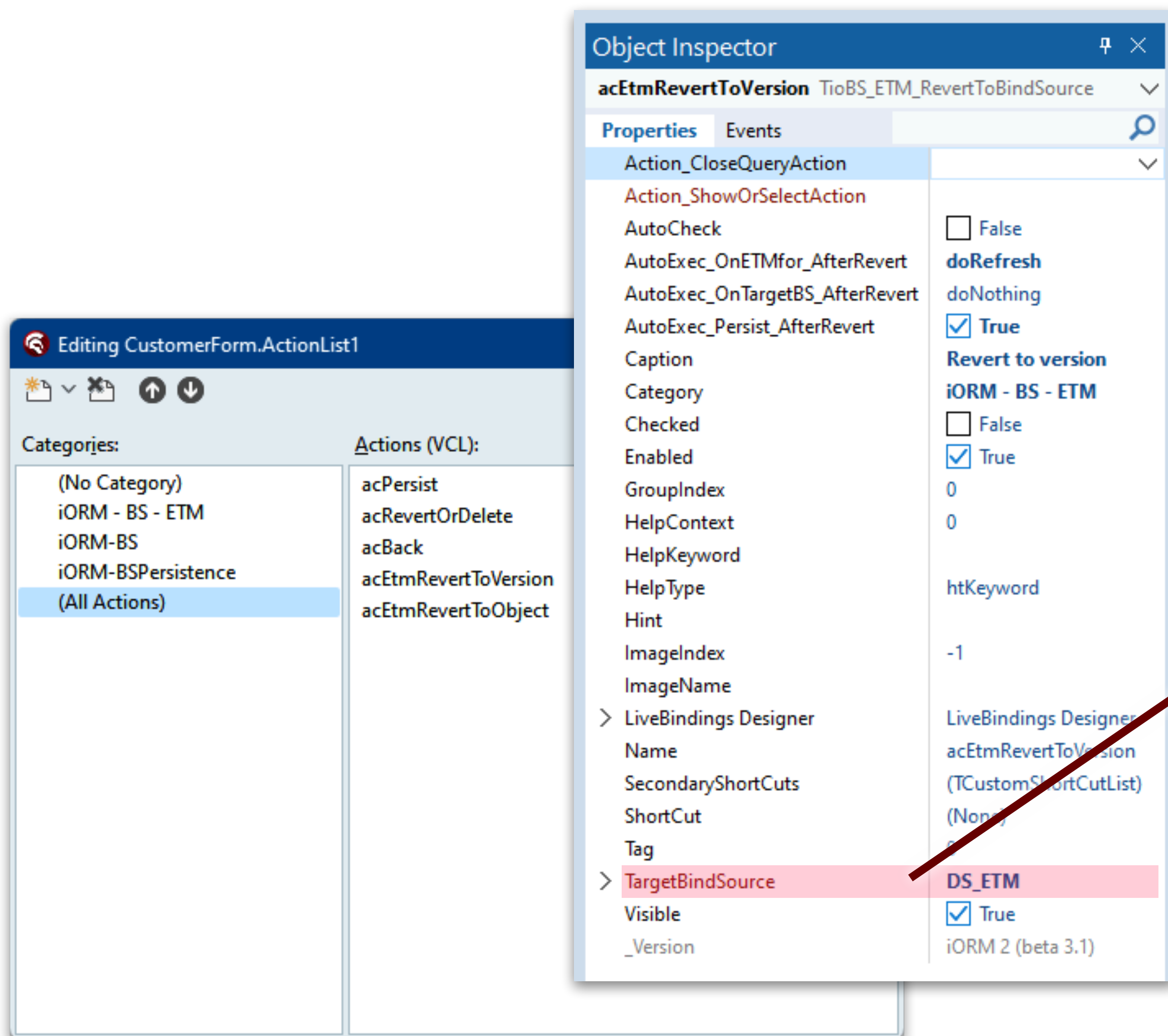
- (No Category)
- iORM - BS - ETM
- iORM-BS
- iORM-BSPersistence
- (All Actions)

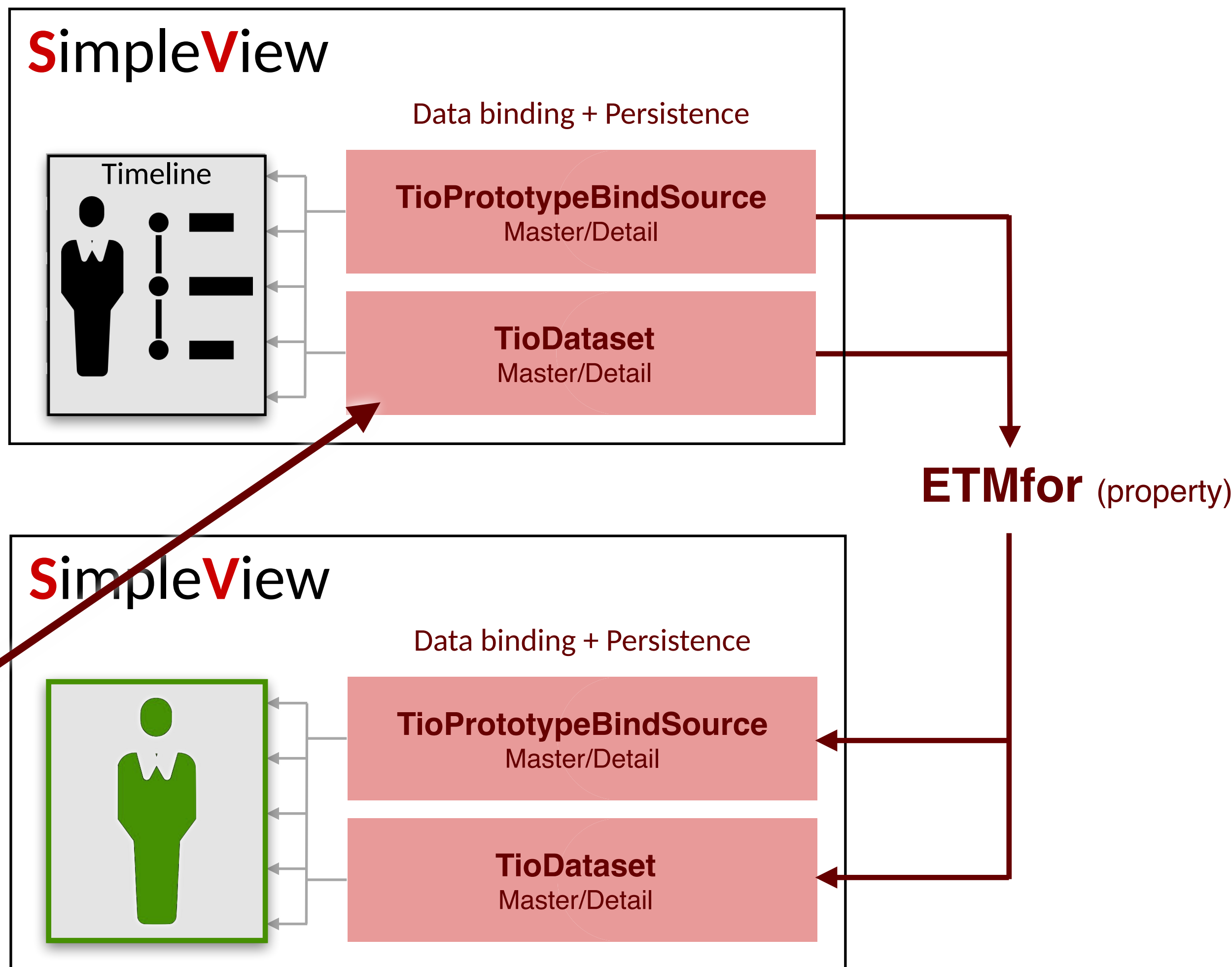
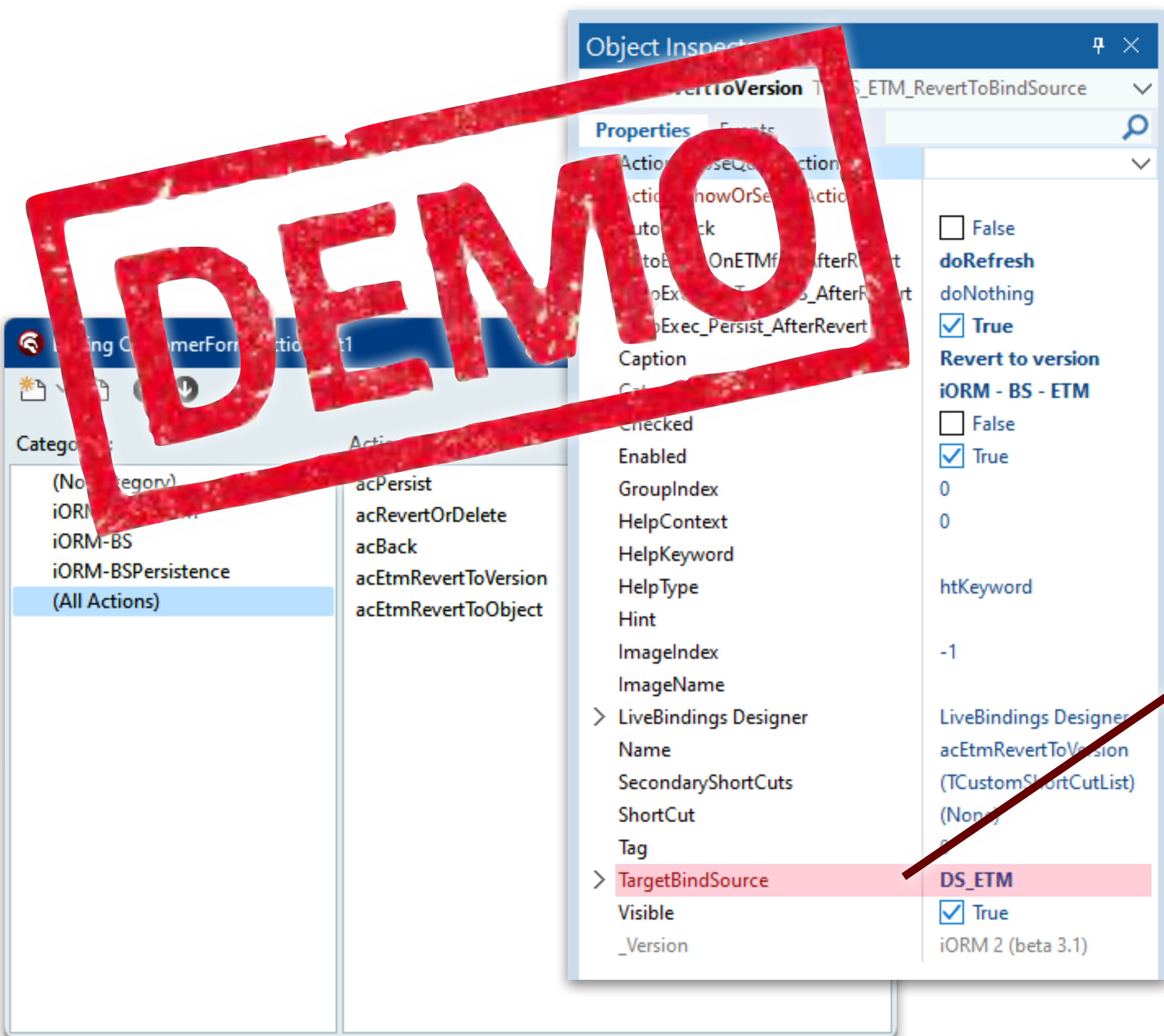
Actions (VCL):

- acPersist
- acRevertOrDelete
- acBack
- acEtmRevertToVersion
- acEtmRevertToObject









Conflict Strategies

an **iORM** feature





ConflictStrategies

an*i*ORMfeature

- Set di **classi** che implementano strategie di **rilevamento e risoluzione dei conflitti**
 - TioSameVersionWin
 - TioLastUpdateWin
- **Attribute** per decorare le classi che si vogliono gestire
 - [ioConflictStrategy]
- **Invocazione automatica** da parte delle persistence strategies
- Possibilità di creare **strategie custom** ereditando da **TioCustomConflictStrategy**



ConflictStrategies

an iORM feature

```
unit iORM.ConflictStrategies.Interfaces;
```

```
...
```

```
// Base class for all ConflictStrategies
```

```
TioCustomConflictStrategy = class abstract
```

```
public
```

```
// This method return a name for this conflict strategy, by default it returns the type name of the class itself but you can override it  
// and return a more readable name. It is used for logging purposes or similar
```

```
class function Name: String; virtual;
```

```
// Check/detect (or prepare the "query") if there is a conflict persisting the DataObject contained into the context
```

```
class procedure CheckDeleteConflict(const AContext: IioContext); virtual;
```

```
class procedure CheckInsertConflict(const AContext: IioContext); virtual;
```

```
class procedure CheckUpdateConflict(const AContext: IioContext); virtual;
```

```
// If a conflict is detected then this method is called from the persistence strategy to try to resolve the conflict
```

```
// Note: the conflict strategy MUST RESOLVE the conflict or raise an exception
```

```
class procedure ResolveDeleteConflict(const AContext: IioContext); virtual;
```

```
class procedure ResolveInsertConflict(const AContext: IioContext); virtual;
```

```
class procedure ResolveUpdateConflict(const AContext: IioContext); virtual;
```

```
end;
```



Conflict Strategies

an iORM feature

```
unit iORM.CommonTypes;
```

```
...  
TioPersistenceConflictState = (csUndefined, csResolved, csRejected, csRejectedRaise);  
...
```

```
[ioEnt...OMERS')]  
[etm...sitory)]
```

```
... (TioLastUpdateWin, csRejectedRaise)]
```

One default conflict state value for all
(Insert, Update, Delete)

```
TCust
```

```
...
```

```
end;
```

Update default conflict state

Insert default conflict state

Delete default conflict state

```
[ioEnt...OMERS')]  
[etm...sitory)]
```

```
... (TioLastUpdateWin, csResolved, csRejected, csRejectedRaise)]
```

```
TCust
```

```
...
```

```
end;
```




ConflictStrategies

an iORM feature

```
[ioEntity, 'RS')]  
[etmTrajectory]  
[ioStrategy(TioSameVersionWin, csResolved)]  
[ioUpdateStrategy(TioLastUpdateWin, csRejected)]  
[ioDeleteStrategy(TioSameVersionWin, csRejectedRaise)]  
TCustomer =  
...  
end;
```

DEMO

Synchronization Strategies

an **iORM** feature





Synchronization Strategies

an **iORM** feature

- Set di **classi** che implementano strategie di **sincronizzazione**
 - **TioEtmSynchroStrategy_Client** (component)
 - **TioEtmSynchroStrategy_Server** (component)
 - **TioEtmSynchroStrategy_Payload** (internal use)
 - **TioEtmSynchroStrategy_LogItem** (it's an entity)
- Possibile **customizzazione** (derivando da)
 - **TioCustomSynchroStrategy_Client** (abstract)
 - **TioCustomSynchroStrategy_Server** (abstract)
 - **TioCustomSynchroStrategy_Payload** (abstract)
 - **TioCustomSynchroStrategy_LogItem**

(All declared in the unit: iORM.SynchroStrategy.Custom)



Synchronization Strategies

anORMfeature

- **TioEtmSynchroStrategy_Client** (component, principali proprietà)

- Async (published)
- Entities (published)
- Entity (published)
- EtmTime (public)
- EtmTime (public)
- TargetConnection (porta alla connessione con server)

- **Connessione locale**

- Innesca impostata



Synchronization Strategies

anORMfeature

- **TioEtmSynchroStrategy_LogItem** (proprietà)

```
// Properties declared in this class
```

```
CliToSrv_TimeSlotID_From
```

```
CliToSrv_TimeSlotID_To
```

```
SrvToSrv_TimeSlotID_From
```

```
SrvToSrv_TimeSlotID_To
```

```
// Properties inherited from TioCustomSynchroStrategy_LogItem
```

```
ID
```

```
SynchroLevel
```

TioSynchroLevel = (slIncremental, slFull);

```
SynchroName
```

```
SynchroStatus
```

```
// User data
```

```
UserID
```

```
UserName
```

TioSynchroStatus = (ssInitialization, ssLoadFromClient, ssPersistToServer, ssReloadFromServer, ssPersistToClient, ssFinalization, ssCompleted);

```
// Counters
```

```
CliToSrv_Count
```

```
SrvToCli_Count
```

```
// Timing
```

```
Start
```

```
LoadFromClient
```

```
PersistToServer
```

```
ReloadFromServer
```

```
PersistToClient
```

```
Finalize
```

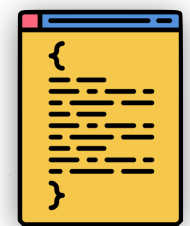
```
Completed
```



Synchronization Strategies

an iORM feature

Avvio della sincronizzazione



bycode (TioCustomSynchroStrategy_Client)

procedure DoSynchronization(const ASynchroLevel: TioSynchroLevel);

TioSynchroLevel = (slIncremental, slFull);



byStdAction

- **TioDoSynchronization** (vcl+fmX)
- **TioVMDoSynchronization** (mvvm)

// some published properties

Autoexec_Enabled

Autoexec_Interval

Autoexec_StartDelay

SynchroLevel

TargetSynchroStrategy

Synchro strategy to be executed

DEMO



MAURIZIO DEL MAGNO DEVELOPER



Lev@nte software



i-ORM

github.com/mauriziodm/iORM

DJSON

github.com/mauriziodm/DJSON



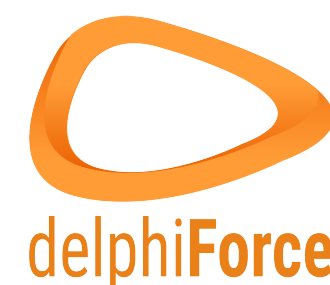
mauriziodm@levantesw.it

mauriziodelmagno@gmail.com



facebook.com/maurizio.delmagno

iORM + DJSON (group)



Membro fondatore

eInvoice4D

<https://github.com/delphiforce/eInvoice4D>



**Multiutenza+Persistenza+Sincronizzazione
=Conflitti**

Conflitti, solo quando sincronizzi due DB?
e quando persisti oggetti ottenuti da una API REST?

