Università degli Studi di Bologna
Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Scuola di Ingegneria

Corso di

Reti di Calcolatori M

COM e DCOM, e .NET

Antonio Corradi

Anno accademico 2014/2015

ARCHITETTURA COM

Architettura Component Object Model e DCOM

WindowsXX famiglia di **sistemi operativi** associata ad una **interfaccia grafica di gestione delle finestre (1985)**

Le prime versioni sono poco più che prototipi con forti limitazioni di uso del processore**: *modalità reale 8086 (limite a 640k di memoria) o modalità protetta 80286***

Windows 3.0 esce nel 1991 e sfrutta le capacità dei processori di classe 386 insieme con le prime applicazioni commerciali di buon livello

Famiglia di ambienti, indicati come WinXX

Windows 3.x (16 bit), Windows 95/98 (ibrido 16/32),
Windows NT (32 bit) e Windows CE (32bit per embedded)
Windows 2000, XP, 2003, eccetera, con nuove direzioni

.NET come nuova architettura

ARCHITETTURA COM

Una serie di sistemi operativi ed ambienti di supporto in crescita come caratteristiche e funzionalità

Windows 3.x (16 bit) non multithreading e solo con divisione di tempo tra processi la cui interazione è guidata dalle finestre

Windows 95/98 ambiente ibrido (parti a 16 bit e parti a 32 bit) orientato a fornire la massima **compatibilità** con il **DOS,** con problemi intrinseci di fragilità

Windows NT per uso professionale con architettura microkernel 32bit molto robusta

Dalla release 4.0 propone **due versioni:**

server e **workstation**

ARCHITETTURA - GUI e DLL

WinXX è un framework orientato alla gestione di finestre e grafica GUI (Graphical User Interface)

costruito intorno al **concetto di finestra visto come base della interazione** e NON SOLO come semplice elemento grafico

tra processi la cui interazione è guidata dalle finestre

Struttura modulare del supporto

il sistema operativo è formato da una collezione di librerie ad aggancio dinamico

DLL (**D**ynamic **L**inked **L**ibraries)

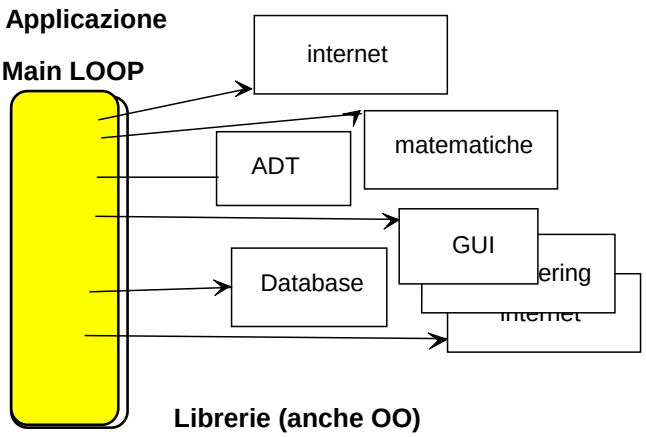
Application Program Interface

le DLL costituiscono la API del sistema operativo

ARCHITETTURA - DLL

Le librerie ad aggancio **dinamico** o DLL (**D**ynamic **L**inked **L**ibraries) sono librerie *con **BINDING DINAMICO** e **NON** SVILUPPATE e **NON** COLLEGATE staticamente con i programmi che le usano*

Una libreria DLL è **sviluppata separatamente** e viene **caricata** e **collegata** solo al momento della prima **richiesta**



Principio di riuso: una **unica immagine** per usi **multipli** e per **processi anche diversi che la condividono**
STATO nella DLL (vincolo di non averlo)

ARCHITETTURA - DLL

WinXX come API distinte per

Win16 (per sistemi a 16 bit) e Win32 (per sistemi a 32 bit)

Il cuore dell'API è formato da **3 DLL:**

Kernel	memory manager, scheduler, loader
User	sistema di windowing
GDI	graphics device interface

Estendibilità

ottenuta **aggiungendo nuove DLL**

per esempio: WinSocket per TCP/IP aggiunte come una libreria dinamicamente caricata al bisogno (una volta sola per ogni nodo)

Si potrebbero caricare altre API di comunicazione

ARCHITETTURA - PROCESSING

Modello **pseudo-multitasking** o “cooperative multitasking”

Win16 ➔ un **solo processo** e **flusso operativo sequenziale**
Il dispatching dei messaggi simula la concorrenza

Se un’applicazione entra in un loop nella risposta ad un messaggio, il sistema rimane bloccato

In caso di unico processo, si mappano la code di messaggi mappate in memoria

Win32 ➔ multitasking effettivo (preemptive)
ogni applicazione è un processo e ha una coda di messaggi

Uno scheduler assegna quanti di tempo ai processi attivi secondo priorità

Ogni processo può avere più thread (processi “leggeri” con memoria condivisa) e, a secondo del sistema, anche buona separazione tra i processi leggeri (WinNT)

In caso di processi distinti, si usano code di messaggi mappate in memoria DDE (Dynamic Data Exchange) ☹

ARCHITETTURA - OGGETTI

Tutte le entità dette oggetti

oggetti utente e **oggetti di kernel**

possiedono un **nome unico** (**handle**) per riferirli e identificarli

con relative **funzioni** di uso (API)

Interfaccia di invocazione non troppo compatta

(*a confronto con UNIX*)

La capacità di esecuzione è attribuita a **processi leggeri** (thread) che agiscono anche in modo coordinato nell'accesso a risorse (spesso inclusi nell'ambito di applicazioni che li racchiudono)

Da WinNT ogni processo pesante può racchiudere un ambiente comune per i thread

ARCHITETTURA - EVENTI MICROSOFT

WinXX introduce il modello ad eventi locali per le interazioni di basso livello a supporto delle finestre

Architettura guidata da eventi callback

"Don't call me I call you" (Hollywood Model)

impostazione comune a tutti gli ambienti a GUI

Evento (di basso livello e di sistema) associato ad ogni **azione dell'utente (mouse tastiera)** e ad ogni **interazione con una periferica (arrivo di un carattere sulla porta seriale o su una socket)**

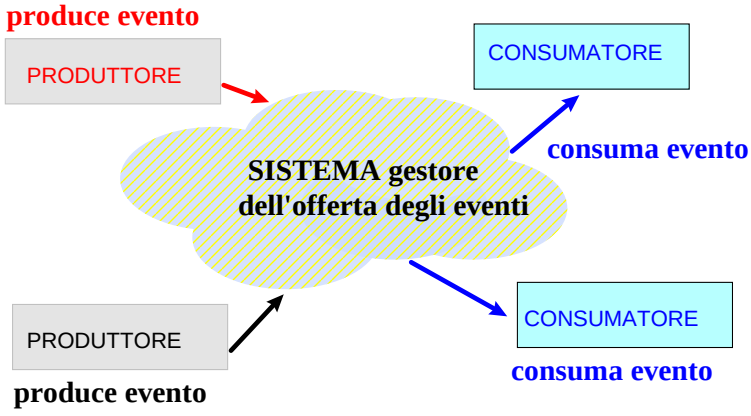
Ogni **evento** provoca l'invio di un **messaggio** ad una o più applicazioni che ricevono i messaggi attraverso una coda

- Win16 **una coda unica** per tutte le applicazioni
- Win32 **una coda separata per ogni** applicazione (o processo)

ARCHITETTURA WIN - EVENTI

Il modello ad eventi asincrono e disaccoppiato, prevede **produttori** e **consumatori** anche non presenti insieme

- il supporto deve occuparsi della gestione per supportare il accoppiamento tra le entità interagenti e garantire la QoS



Gestione della maggior parte dei sistemi a finestre **push** dell'evento alle entità interessate e **registrate**

Uso di **code dei messaggi** per i **processi** dette **DDE** o **Dynamic Data Exchange**

ARCHITETTURA - EVENTI LOCALI

Messaggi come base della comunicazione fra applicazioni e fra finestre di un'applicazione

In Windows una finestra è un'entità in ***grado di ricevere e elaborare messaggi accodati da eventi***

L'applicazione attraverso il mouse passa un codice che specifica la funzione di gestione richiesta (puntatore ad una funzione) alla DLL della finestra

Meccanismo di callback consente ad una DLL di sistema di richiamare una funzione della applicazione

Implementazione del sistema di **messaggi/eventi**

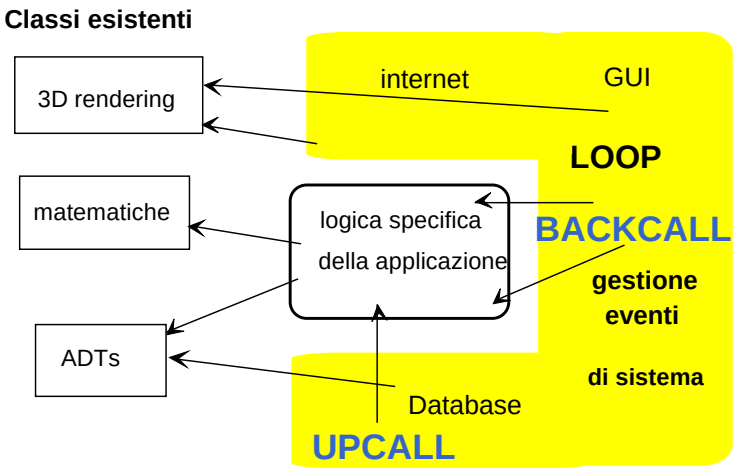
Ogni finestra è associata ad una funzione di **callback** (Window Procedure o WndProc) di un'applicazione usata dalle DLL di sistema per inviarle i messaggi

La ***funzione di Callback locale*** *permette di associare una gestione* ***specificata dall'utente all'evento****, lasciando molte azioni al sistema che le gestisce a default*

ARCHITETTURA - EVENTI

Un programma registra una callback per ogni evento e per ogni finestra

Quando la finestra è a fuoco e facciamo una azione, mandiamo un evento alla applicazione stessa



Al contrario di normali librerie le invocazioni non arrivano solo dall'alto ma anche dal sistema di gestione (framework)
UPCALL /BACKCALL

ARCHITETTURA a COMPONENTI

Componenti come entità capaci di fornire servizi in modo pervasivo
dovunque, per chiunque, in ogni momento
senza limiti alla riusabilità e **con inserimento dovunque**

Microsoft ha interesse di consentire il progetto di componenti autocontenuti, indipendenti dal linguaggio, capaci di interagire tra loro in modo libero, e senza legami predefiniti con un ambiente di programmazione

Si noti che i **componenti** sono più **pervasivi degli oggetti**, in quanto gli oggetti tendono ad avere regole di invocazione più precise e legate spesso ad un ambiente di linguaggio

Si noti che i componenti spesso sono legati alla idea di un contenitore che permette di associare al componente molte funzionalità a default

Modello di interazione tra componenti (e anche con il contenitore)

DOCUMENTI COMPOSTI (OLE)

Object Linking & Embedding o OLE

Integrazione di componenti provenienti da fonti diverse all'interno di un **documento composto**

per dividere le responsabilità di gestione

Container: applicazione che gestisce il documento

Server: applicazioni specializzate che gestiscono i componenti

Linking: il documento contiene solo un riferimento al componente

Embedding: il componente viene incorporato nel documento

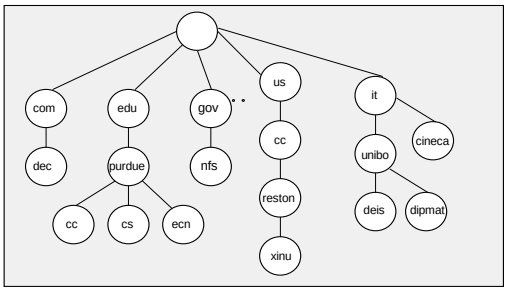
In-place activation: l'applicazione server si attiva nello spazio visivo (finestra) del container su richiesta (outside-in)

Object Linking & Embedding

OLE per una gestione organizzata ed efficace

In un file powerpoint oggetti powerpoint, ma anche oggetti diversi, come un oggetto word con la sua gestione

Spesso organizzazioni a livelli
gestore generale che coordina gestori di più basso livello in molti livelli con località informazioni (vedi CORBA o DNS)



La parte di documento a sinistra è gestita da Word e può essere composta di sottoparti

OLE per APPLICAZIONI INNESTATE

OLE è la tecnologia **base Microsoft** con cui **si può attivare una applicazione all'interno di un'altra**

In un documento Word con una tabella Excel, la tabella continua a riferire il proprio gestore e lo attiva se e quando necessario

Problemi di **duplicazione** o **riferimenti** allo stesso oggetto

Riferiamo l'oggetto specificato (riferimento) o

ne facciamo una copia (valore)

Problemi sull'**insieme di gestori associati allo stesso documento da attivare e del loro costo**

Attiviamo con l'oggetto specificato il gestore principale o anche tutti gli altri che possono essere riferiti da tutte le sue parti

MIDDLEWARE MICROSOFT

Necessità di supportare un insieme di componenti disponibili per diverse applicazioni e **capaci di implementare funzionalità** in modo **differenziato**

- COM e i successivi supporti prevedono componenti a **grana fine**, in termini di **interfacce** possibili: i componenti posso anche implementare molti comportamenti, cioè **molte interfacce**
  - La scelta tra le interfacce è tipicamente a **carico del sistema** e scelta in base a strategie differenziate: il middleware è caricato su bisogno e anche i suoi componenti
- Le applicazioni, tipicamente **clienti** ma anche **servitori** hanno un **tempo di vita**: mantenute in uso finché servono; attivazione e disattivazione **gestite** in '**modo assistito**'
  - La scelta del **componente** per implementare una interfaccia è **ottimizzato** per il migliore adattamento

MIDDLEWARE per APPLICAZIONI VELOCI

Lo **scenario COM/DCOM** è quello di un **middleware** per **applicazioni ottimizzate**, che richiedono servizi e li ottengono

- Le applicazioni possono anche fornire **servizi reciprocamente**; il **middleware** usa sia le **funzioni proprie** sia quelle disponibili nella **situazione corrente** in modo dinamico
- Il middleware facilita ma **non abilita** (supporta) ogni operazione:
- la **interazione** tra entità è **peer-2-peer** (diretta e non **mediata**)

Middleware per interazione applicazioni

- I middleware Microsoft sono i primi a pensare ad una logica di supporti estensivi su architetture in crescita (in evoluzione): **forte accento sulla compatibilità**
- Supporto alla **dinamicità** e **anche possibili scelte ottimizzate in modo automatico o su indicazione utente**
- **Tempo di vita del middleware** legato alle **applicazioni** e alle loro **strategie**

ARCHITETTURA COM - NOMI

Necessità di avere un sistema di nomi non ambiguo

COM si basa su un **sistema di nomi** per i componenti

utilizzando identificatori globali

GUID=Globally Unique Identifiers

{32bb8320-b41b-11cf-a6bb-0080c7b2d682}

GUID intero di 128 bit (16 byte) assegnato per garantire l'unicità nello spazio e nel tempo

Si usano nomi unici per oggetti (**GUID**), classi (**CLSID**), e interfacce (**IID**)

UNICITÀ dei NOMI

GUID introdotto dalla **Open Software Foundation** come **UUID** 'Universally Unique ID' nel sistema delle RPC di DCE (**Distributed Computing Environment**)

ARCHITETTURA COMUNICAZIONE

Component Object Model COM e Distributed DCOM

Come mettere insieme le cose? (1992) **COM**

modello di **interazione fra applicazioni, processi e librerie**, basato su

componenti e

comunicazione client/server sincrona

COM è un insieme di specifiche supportato da servizi di sistema (supporto runtime come la DLL di sistema per i servizi di accesso ai componenti)

Le specifiche definiscono lo standard (Microsoft) per la creazione di componenti in grado di interagire fra loro

in modo indipendente

*dal **linguaggio di programmazione** e*

*dalle **applicazioni** che li utilizzano*

ARCHITETTURA COM - DCOM

Component Object Model e Distributed DCOM

WinXX come **architettura multilinguaggio** ma **proprietaria** per la **comunicazione ottimizzata tra componenti indipendenti (logica di middleware peer-to-peer)**

La versione distribuita adotta comunicazione standard

Uso delle RPC di DCE (**D**istributed **C**omputing **E**nvironment)

Il modello Winxx identifica uno scenario di funzionamento omogeneo tra **componenti** per ambienti Winxx

- **locali** e
- **distribuiti** (*basato su RPC standard*)

basato su un unico modo di interagire con obiettivo di ottimizzazione

ARCHITETTURA COM - INTERFACCE

Component Object Model (*COM*) *creato per C++*

un oggetto **COM** viene identificato univocamente da un **GUID** ed espone una o più **interfacce**

Un'**interfaccia** è l'insieme di tutte e sole le **funzioni** (***metodi***) che permettono di **interagire con l'oggetto** che la espone (come in Java)

In genere, le **interfacce nascondono i dettagli dell'oggetto** ed sono **decise prima della esecuzione**

il meccanismo delle interfacce nasce dalla tecnica usata per oggetti definiti e istanziati in DLL

Un oggetto mostra un insieme statico di interfacce (ogni componente ha molte interfacce, granularità piccola)

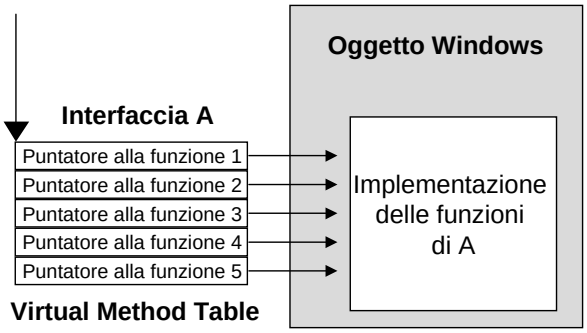
Se ogni componente può essere identificato da un **GUID**
le **interfacce** sono identificate da **GUID** chiamati **IID** (**I**nterface **ID**entifiers)
Per convenzione i **nomi logici delle interfacce** iniziano con la lettera **I**

*Ad esempio la interfaccia universale **IUnknown***

ARCHITETTURA COM - INTERFACCE

Ogni Oggetto COM deve esporre almeno una *interfaccia generalissima* detta ***IUnknown***

Questa interfaccia è la base per **la comunicazione** dei componenti nelle applicazioni e per **gestire** il tempo di vita



Implementativamente

Interfaccia come **puntatore** ad una **porzione** di "**virtual method table**" (*vtable o VMT*) di un oggetto, cioè gli accessi ai suoi metodi

La **VMT** *è una tabella di* ***puntatori a funzione*** *del componente che consente di ritrovare poi i diversi metodi invocabili sullo stesso componente*

COM – INTERFACCE MULTIPLE

Ogni **Oggetto COM** espone **molte *interfacce***

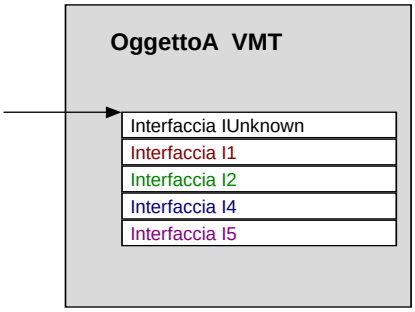
Sicuramente, per essere un componente DCOM, ***IUnknown***

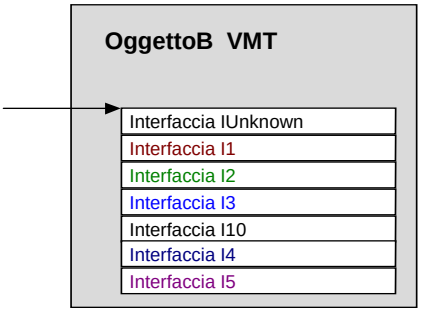
poi una **serie di altre interfacce**, che dipendono da quante l'oggetto ne vuole fornire all'utilizzatore esterno

Un componente ha una propria VMT come una sequenza di interfacce: come le interfacce sono spiazzate nella specifica VMT dipende da ogni singolo oggetto

Ogni metodo può essere invocato solo conoscendo la posizione della interfaccia nella VMT e il suo spiazzamento rispetto a quella

posizione interfaccia di interesse + spiazzamento metodo





ESEMPIO di INTERFACCIA

```
type IDispatch = interface(IUnknown)
  ['{00020400-0000-0000-c000-000000000046}']
HRESULT GetTypeInfoCount(out Integer Count);
HRESULT GetTypeInfo(Integer Index, LocalID;
     out TypeInfo * ptrInfoArea);
HRESULT GetIDsOfNames(const IID TGUID; Pointer Names;
     Integer NameCount, LocalID; out Pointer DispIDs);
HRESULT Invoke(Integer DispID; const IID TGUID;
     Integer LocalID; Word Flags; out Pointer Params;
     out Pointer VarResult, ExceptInfo, ArgErr);
end;
```

Ereditarietà delle Interfacce: **IDispatch** eredita da **Iunknown**
Interfacce come insieme di **metodi**: **IDispatch** ha quattro metodi
Metodi con **parametri** alla C++: **parametri** anche in uscita o costanti
Tipi: primitivi, HRESULT, Word,*, Pointer, … IID, …
GUID presenti a 128 bit

INTERFACCE di BASE COM

Il famigerato **Registry** è il sistema di **nomi locale** che registra tutti i GUID dei server delle applicazioni locali (vedi applicazione **regedit**)

Nel **registry** ci sono **informazioni dette chiavi**, che possono includere attributi (sottochiavi), ecc. e poi **la locazione del componente** (pathname)

HKEY_CLASSES_ROOT\CLSID che lista gli UUID delle classi

HKEY_CLASSES_ROOT\Interface che lista gli UUID delle interfacce

si consente di trovare e gestire oggetti e interfacce

IUnknown è una sorta di punto di ingresso
per accedere a tutte le altre interfacce esposte da un oggetto
Un oggetto COM espone anche la sua ***IUnknown*** con *IID*
`'{00000000-0000-0000-C000-000000000046}'`

IUnknown comprende 3 metodi
QueryInterface, AddRef e Release

INTERFACCIA di BASE in COM

QueryInterface permette di **accedere alle interfacce esposte dall'oggetto** (che in genere ne espone molte)

```
HRESULT QueryInterface(const GUID REFIID;
                       out void ** ObjectPtr);
```

richiede all'oggetto se implementa *una determinata interfaccia con parametro REFIID*
HRESULT intero dice se l'interfaccia richiesta *è disponibile* (S_OK) oppure *no* (E_NOINTERFACE)

In caso positivo, il parametro **`ObjectPtr`** **contiene *un riferimento all'interfaccia richiesta*** e permette di usarlo per richiederne i metodi

AddRef *e* ***Release*** *per "reference counting"*

Ogni volta che un componente viene attivato si incrementa il reference count e ogni volta che qualche altro lo scarica si decrementa

A zero si scarica il componente

Gestione tempo di vita di oggetti COM

IUnknown permette anche di **gestire il tempo di vita degli oggetti COM,** usando il reference counting, cioè un intero a 32bit associato all'oggetto stesso *(AddRef* e *Release)*

La deallocazione degli oggetti può avvenire in diversi modi:

- **nessuna gestione** (*oggetti restano anche se non usati*)
- **garbage collector** IMPLICITO
  (*oggetti restano solo se risultano ancora riferibili da altri*)
  in un sistema distribuito può essere problematico e costoso
- **reference counting** ESPLICITO
  (*oggetti risultano ancora riferibili se refcount positivo*)
  gli utenti devono gestire la accessibilità dei riferimenti

INTERFACCE in COM

COM supporta **ereditarietà** delle (sole) **interfacce**

(**non** delle implementazioni)

Si stressa la programmazione di interfacce più che di oggetti

Ereditarietà delle interfacce

Tutte le interfacce COM discendono da ***IUnknown***

tutte le VMT corrispondenti hanno i metodi *QueryInterface, AddRef e Release* nei primi 3 slot della VMT (*fissi e non sovrascrivibili*)

L'ereditarietà delle interfacce è usata pervasivamente: tutte le interfacce discendono da ***IUnknown*** o da ***IDispatch*** (che discende ed eredita da ***IUnknown***)

RIUSO in COM (NO EREDITARIETÀ)

COM vede l'**ereditarietà** come un ***accoppiamento troppo stretto*** fra implementazioni degli oggetti e la sconsiglia a livello di MW

ereditarietà è vista come una pericolosa breccia nell'incapsulamento

una classe derivata può accedere allo stato interno della classe base

Ogni **componente** deve essere **auto-contenuto e protetto**

COM propone un **meccanismo di riuso alternativo** basato su due meccanismi detti **aggregazione e delegazione**

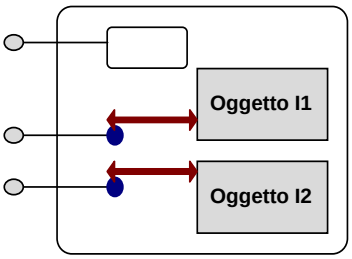
➔ un oggetto espone anche interfacce implementate non direttamente ma da altri oggetti detti proxy (**aggregati** - se esterni - o **contenuti** - se interni)

➔ comunque si richieda la operazione, la richiesta:

- viene trasferita *implicitamente* agli oggetti **aggregati** (esterni) che rispondono autonomamente
- viene passata *esplicitamente* all'oggetto **delegato**/**contenuto** in quello esterno e **dipendente** dalla attivazione dell'oggetto esterno stesso

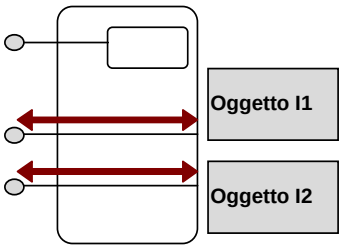
AGGREGA- DELEGA- ZIONE in COM

Oggetto Interno



Delegazione/Contenimento

Oggetto Esterno



Aggregazione

Aggregazione e Delegazione diverse dalla ereditarietà

Ogni metodo deve essere agganciato alla sua implementazione come nei sistemi prototipali ad attori

*Molto **meno eleganti e più complesse dell'ereditarietà** (poco più di una specifica di implementazione)*

In caso di ereditarietà le classi ereditano i metodi e **'implementano'** solo i metodi in proprio (che aggiungono)

AGGREGAZIONE in COM

Aggregazione (oggetti indipendenti)

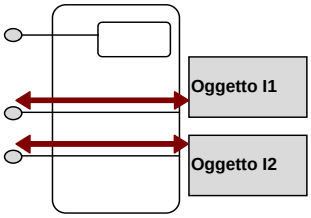
La interfaccia è suddivisa tra i diversi oggetti che ne sono responsabili e permette anche di fare una gestione separata ed efficace

Raccomandata per una possibile **gestione a parti**

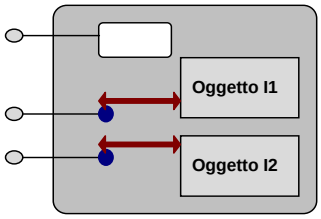
Delegazione (oggetti interni ad altri)

L'entità contenitore presenta la interfaccia unica e deve essere presente attivamente per passare le richieste ai *contenuti*

Problemi di gestione degli oggetti e del loro tempo di vita in caso di innestamento e relazioni implicite tra oggetti



Aggregazione



Delegazione Contenimento

In COM la configurazione iniziale è complessa e deve essere aiutata da wizard

‘FILOSOFIA’ COM

- **Componenti** come **entità fondamentali e nessuna ereditarietà della implementazione**
- **Componenti** che implementano diverse **(molte) interfacce** che possono essere anche ad **ereditarietà multipla tra loro**
- **Componenti** che possono al loro interno avere **altri componenti (in modo ricorsivo)** o **aggregati** al loro esterno
- **Creazione dei componenti specificata** e anche **loro distruzione** sotto il **controllo dell’utente**
- **Uso di riferimenti remoti non persistenti** per raggiungere i componenti ed ottenere servizi

- Una applicazione carica i **propri servitori e clienti** e comincia ed eseguire su più macchine che sono state inizializzate in COM
- Anche presenza di **più applicazioni** contemporaneamente

C/S in COM: CLIENTE

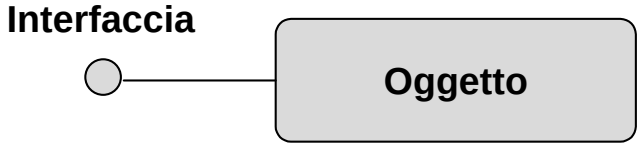
Un oggetto o componente può chiedere una operazione mostrata dalla interfaccia di un altro oggetto **COM**

***Indipendenza dalla collocazione fisica*: l’applicazione** che usa un oggetto **COM (client) non sa dove risiede** effettivamente **l’oggetto server** ma accede in modo **omogeneo alla interfaccia per i metodi**

Server: entità che rappresenta un oggetto COM che possa essere invocato da altri oggetti, tramite l’intermediazione della interfaccia corrispondente (IUnknown)

Middleware: dirige verso il miglior corretto componente

Rappresentazione semplificata di INTERFACCIA



FACTORY per il SERVITORE

Un oggetto per essere un server **COM** deve potere essere gestito da una **Factory (oggetto ad-hoc)** in grado di attivarlo e mantenerlo

Factory (non classe) come ***componente per gestire oggetti:*** per ***creare*** (attivare) ***/ distruggere*** (disattivare)

Ogni server deve specificare una interfaccia *IClassFactory* che permette alla Factory di gestire gli oggetti server

> **In alternativa, l'Interfaccia *IClassFactory2* permette una creazione tenendo conto di licenze degli oggetti stessi**

Il server deve registrare nel registry locale il CLSID della classe implementata e il percorso per raggiungerla (nel file system)

La registrazione permette di avere un accesso alla implementazione per il caricamento/scaricamento

Class Factory: IClassFactory

IClassFactory permette alla Factory di **attivare istanze e gestirle**

```
type IClassFactory = interface(IUnknown)
HRESULT CreateInstance(IUnknown * pOuterComponent,
      GUID REFIID, out void ** ObjectPtr);
HRESULT LockServer(BOOL fLock);
end;
```

In alternativa anche la ***CreateInstance*** attiva una **istanza del tipo richiesto** (usando appunto la Factory per la creazione) o **aggancia una istanza già attivata**

> Viene anche usata per la aggregazione e fornisce un'interfaccia per il contenitore esterno (se esiste); ObjectPtr è l'interfaccia dell'oggetto creato

La ***Lockserver*** permette di stabilire un comportamento per la Factory che potrebbe **mantenere il server in memoria** con reference count a zero

> In questo caso, *le risorse vanno mantenute e il cliente specifica il lock per evitare di non trovare più la factory stessa*

Class Factory: IClassFactory2

IClassFactory2 permette di **lavorare con licenza (file testo)**

```
type IClassFactory2 = interface(IClassFactory)
HRESULT CreateInstanceLic(IUnknown * pOuterComponent, IUnknown *
    pUnkReserved, GUID REFIID, BSTR bstrKey, out void ** ObjectPtr);
HRESULT RequestLicKey (DWORD dwReserved, BSTR * pbstrKey);
HRESULT GetLicInfo (LICINFO * pLicInfo);
end;
```

La ***CreateInstanceLic*** crea una istanza del tipo richiesto (su una macchina con licenza appropriata) con una chiave che possa essere usata successivamente dal cliente per richiedere ulteriori servizi

La ***RequestLicKey*** permette di ottenere una chiave per richiedere operazioni protette da licenza

La ***GetLicInfo*** permette di stabilire una struttura per determinare la licenza per la Factory

LIBRERIA COM - CLIENTE

Ci sono un set di funzioni accessorie (vista la difficoltà)

CoBuildVersion	fornisce la versione della libreria COM
CoInitialize	inizializza il supporto COM
CoUnitialize	scarica un cliente dal supporto COM e rilascia tutte le risorse

La elaborazione per un cliente

CoCreateInstance ottiene l'oggetto server per eseguire una necessaria operazione

```
HRESULT CoCreateInstance (CLSID ReqCLSID, IUnknown * pOuterComp,
    DWORD ClassContext, GUID REFIID, out void ** ObjectPtr);
```

determina un oggetto della classe specificata e restituisce **un puntatore** alla istanza (da usare per richiedere una operazione) *La ricerca avviene nel **registry** e il **contesto** dice dove cercare tra le possibili scelte* (locale: libreria e eseguibili, o anche remoto)

RIFERIMENTI ad OGGETTI

In COM e DCOM

I riferimenti agli oggetti sono non persistenti e lasciati ad una gestione utente che deve occuparsi di dettagli complessi e di difficile gestione

Gli **ObjectPtr** sono **diretti su entità** (dalla rispettiva Factory) che realizzano una specificata interfaccia, poi sono agganciati in modo preciso all'oggetto specifico (e non mobili)

Gli **ObjectPtr** sono entità **ottenute run-time**, si possono passare da un cliente ad un altro, e da preservare e mantenere da parte dei diversi clienti, ma **non sopravvivono** alla applicazione

Gli **ObjectPtr** sono per entità con una **organizzazione di esecuzione complessa**, costituita anche da **oggetti contenuti e contenenti**, che deve essere consistente e safe

La gestione degli ObjectPtr è particolarmente complessa in termini di **mappaggio delle loro relazioni** e di **identificazione dei diversi oggetti** (anche di tipo diverso) da puntare e da mantenere nei diversi ambienti

MONIKERS – NUOVI NOMI AGGREGATI

I riferimenti di DCOM sono **non persistenti** e hanno un **tempo di vita limitato** dalle applicazioni relative

Per **avere flessibilità e durata maggiore**, si introducono nomi che possano sopravvivere oltre la durata del **programma**

i **MONIKER**, ossia

- **nomi persistenti per gruppi di oggetti**
- **indipendenti dalle applicazioni**
- **salvabili nella memoria persistente del file system**

Moniker type	Description
File moniker	Reference to an object constructed from a file
URL moniker	Reference to an object constructed from a URL
Class moniker	Reference to a class object
Composite moniker	Reference to a composition of monikers
Item moniker	**Reference to a moniker in a composition**
Pointer moniker	**Reference to an object in a remote process**

MONIKERS – NOMI CONTESTUALI

i MONIKER riferiscono risorse esistenti e persistenti e le associano a un programma

- i **moniker** risolvono il problema della **creazione** personalizzata e **differenziata**, permettendo di ritrovare comportamenti contestuali
- i **moniker** possono essere memorizzati e contenere un protocollo di **binding interno** e **contestuale**

I moniker permettono di definire un **livello di nomi indiretto** che si possa legare ad un contesto nel quale si possano dare valori ad attributi

FileMoniker, ItemMoniker, PointerMoniker, … URLMoniker

Tipi di moniker differenziati per specificare che alcune proprietà possono variare a secondo delle situazioni e della invocazione

- Quindi lo stesso oggetto riferito viene attivato in modo diverso a secondo di come lo si richiede
- *la interfaccia **IMoniker** lavora a livello dei nomi indiretti e trasparenti alla allocazione*

CLIENTE/SERVITORE in COM

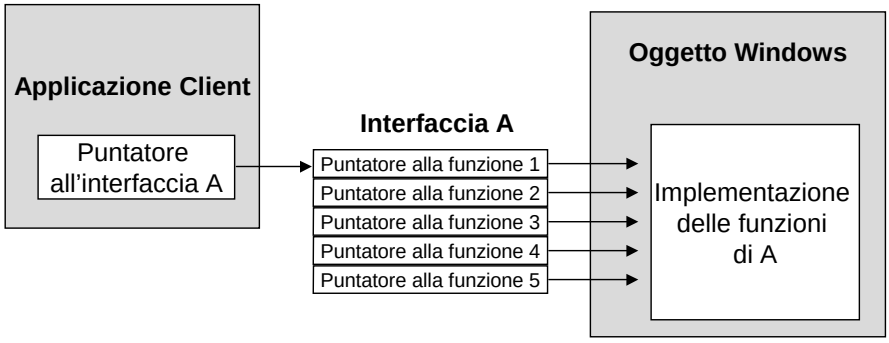
Sono previsti tre tipi di server

In-process server: all'interno dello stesso spazio del processo utente, in una DLL (eventualmente condivisa da altri processi sulla stessa macchina)

Local server: fuori dal processo cliente e sulla stessa macchina (in un EXE separato)

Remote server: in altra macchina DCOM (Distributed COM)

I server locali e remoti vengono detti "out-of-process"



In-process server o DLL

Una DLL esporta quattro funzioni standard: *DllGetClassObject*, *DllCanUnloadNow*, *DllRegisterServer*, *DllUnregisterServer*

DllRegisterServer	registrazione dei componenti della DLL nel registry locale
DllUnregisterServer	deregistrazione dei componenti della DLL
DllCanUnloadNow	richiede alla DLL se si possa scaricare: la DLL deve avere uno stato in grado di verificare la safety del tutto
DllGetClassObject	verifica se la DLL supporta la CLSID richiesta e ne istanzia una Factory per creare l'istanza

```
HRESULT DllGetClassObject (CLSID ReqCLSID, REFIID ReqIntID,
                out void ** ObjectPtr);
```

Più il supporto per gestire le risorse (ossia il tempo di vita della DLL)

LoadLibrary	carica la DLL nello spazio del chiamante
FreeLibrary	il chiamante scarica la DLL
GetProcAddress	trova le funzioni della DLL

In-process server via COMOBJ

L'applicazione cliente richiede un oggetto al supporto runtime (COMOBJ) sulla base delle GUID, IID, e CLSID

Si nascondono alcuni meccanismi di dettaglio

COMOBJ cerca nel registry il nome della DLL associata al GUID e la prepara per il caricamento in memoria (se non già presente)

Chiama una funzione della DLL e ottiene l'interfaccia *IClassFactory* (vtable di un oggetto *ClassFactory*)

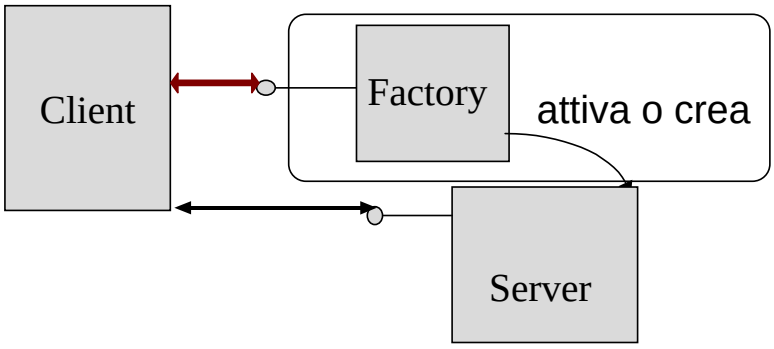
Chiede alla ClassFactory di instanziare l'oggetto richiesto e di restituirne l'interfaccia (o di usarne una già presente in memoria e registrata)

IUnknown (*IClassFactory.CreateInstance*)

L'interfaccia viene passata al cliente che invoca i metodi della DLL (dopo la GetProcAddress)

Local e remote server: ALTRO PROCESSO

Il cliente locale usa la interfaccia ***IUnknown*** restituita dalla Factory



Interazione **sempre sincrona bloccante**

Si noti che la stessa interazione consente di muoversi

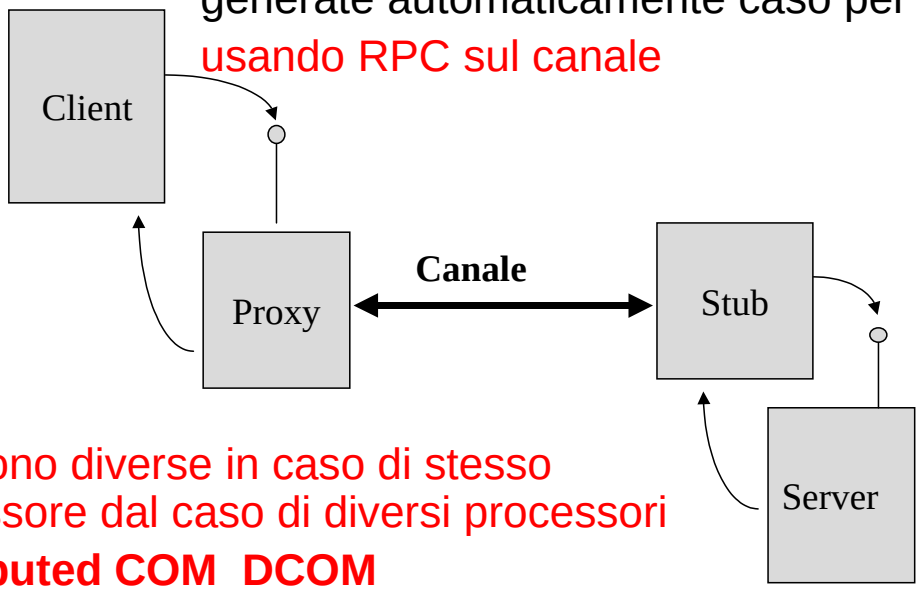
- tra **processi** della stessa applicazione
- tra **applicazioni** della stessa macchina
- tra **processi** su **macchine diverse** DCOM

COM per local e remote process

In caso si **comunichi tra processi diversi**, entrano in gioco della entità intermediarie, dette **proxy** e **stub**

generate automaticamente caso per caso

usando RPC sul canale



NB. Sono diverse in caso di stesso processore dal caso di diversi processori

Distributed COM DCOM

MIDL in COM

Le interfacce vengono descritte con un linguaggio apposito **Microsoft IDL** (**I**nterface **D**efinition **L**anguage)

- Il linguaggio è un **normale IDL** per descrivere dati e formato con una definizione per fornire i supporti allo sviluppo statico
- non compatibile con altri IDL o con altri standard del settore

MIDL è un linguaggio con una sintassi simile a quella del C ed è compatibile con lo standard ***DCE IDL***

- Incorpora un meccanismo precedente (ODL = Object Definition Language) che forniva una soluzione parziale legata ad **OLE** Automation
- **Object Linking & Embedding** (tecnologia precedente di Windows)

PRODOTTI del MIDL

Nel caso si lavori su una **unica macchina**, ci sono meno problemi ⇨ *nessuna eterogeneità*

Ad esempio,

- ***stessa rappresentazione*** *dei dati*
- *più facile passare dalla* ***esecuzione nel cliente***
  a quella ***nel servitore***
- *più facile* **condividere memoria per i parametri**

Un **compilatore MIDL** genera automaticamente la coppia ***proxy/stub*** a partire da una di queste descrizioni

Ne genera una ***coppia per ogni oggetto server e client***

- *Deve essere previsto in modo statico*
- *Per ogni linguaggio*
- *Per ogni coppia*

Una coppia per il caso locale, una coppia per il caso remoto ☹

RICHIESTE di un CLIENTE e SERVIZIO

Un cliente parte da una esigenza specificando una **interfaccia**

Alla richiesta, si cerca un **componente locale** che la implementi nel registry locale

- ⇨ *se esiste, ci si collega o si attiva una **DLL locale**, realizzando un **servizio in-process***
- ⇨ *se esiste, ci si collega o si attiva un **processo locale**, cioè un **server locale***

⇨ se non esiste un componente locale, allora si deve trovare un **componente remoto adatto**

In genere, nel registry sono listati solo componenti locali, ma possono essere conosciute altre allocazioni

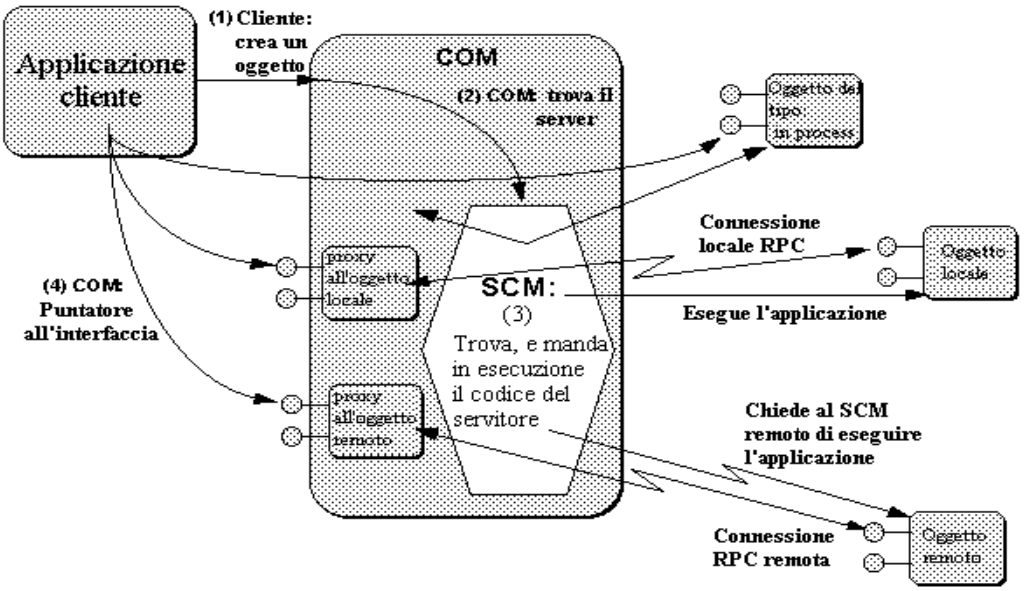
L'utente può suggerire sue allocazioni preferite

O il supporto per servizi remoti deve cercare altre locazioni

⇨ ***ci si collega o si attiva un processo remoto, cioè si comunica con un altro nodo, via RPC***

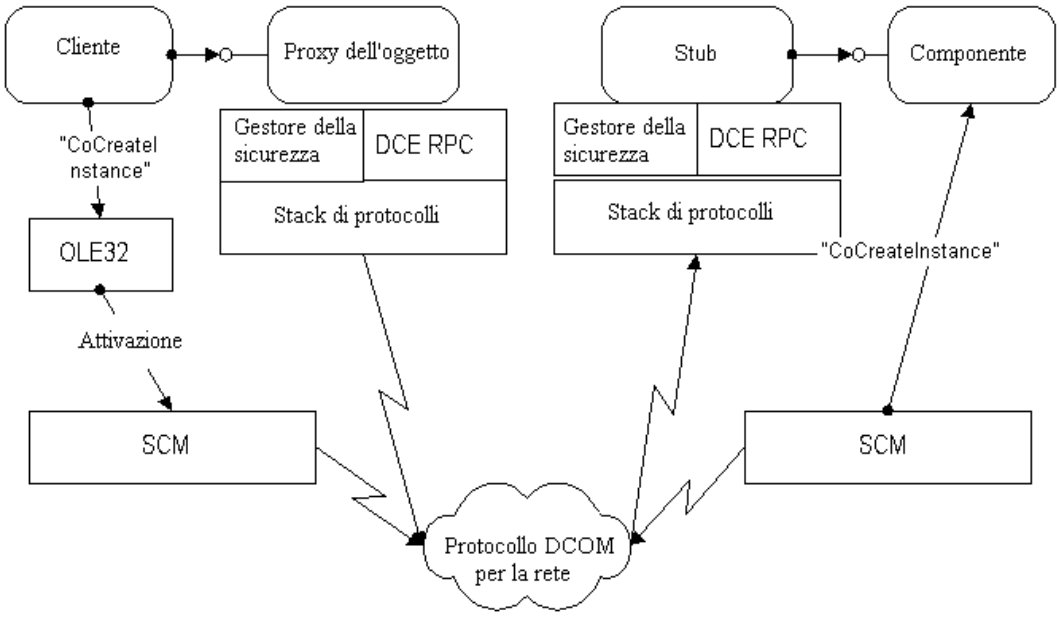
Remote Process in DCOM

SCM (**S**ervice **C**ontrol **M**anager) il componente che interviene nel caso di servitore remoto per gestire la interazione (senza fare naming ma solo come abilitatore della comunicazione)



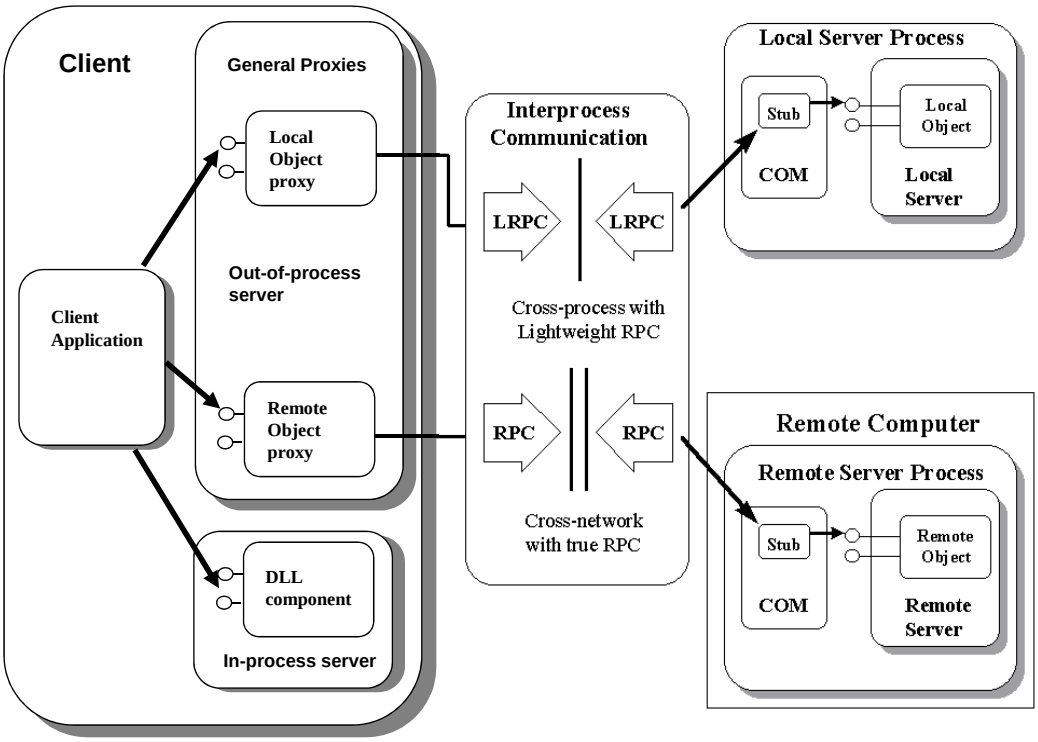
INVOCAZIONI REMOTE DCOM

Si usano tradizionali procedure remote (**RPC** proposte da **DCE**) come veicolo per lavorare internodo



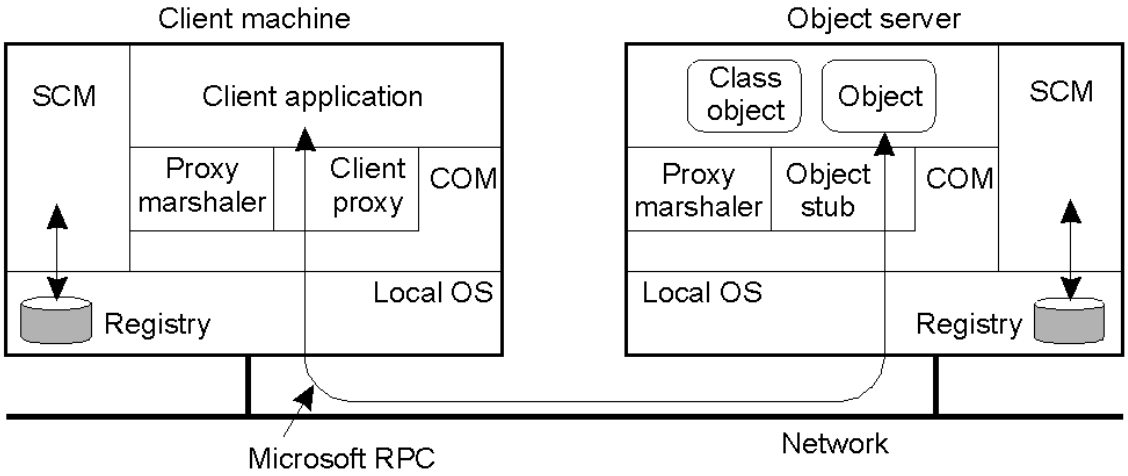
DCOM – OUT-OF-PROCESS

Omogeneo in **caso locale** (local process via *Light RPC*) e in **caso remoto** (processo remoto via *RPC* secondo il modello DCE)



ARCHITETTURA DCOM

Per arrivare da un **cliente** ad un **servitore in modo *sincrono*** sono messi in gioco sia entità di **supporto ad-hoc** (*proxy e stub dalle due parti*) sia **supporto di naming** (***registry*** *e* ***Active Directory o AD*** *nel distribuito, con* ***Service Control Manager*** *per il coordinamento tra nodi*)



ARCHITETTURA COM e DCOM

I middleware COM DCOM si articolano intorno al concetto di **interfaccia** (i componenti ne implementano più di una)

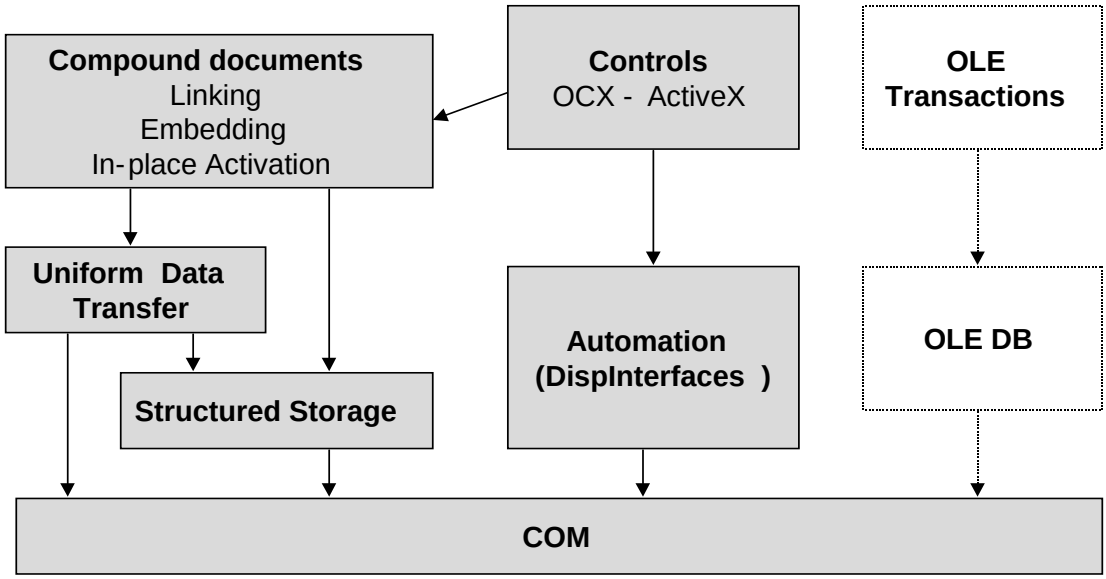
- Le interfacce sono a **grana fine** e sono implementate da **componenti diversi** con diversi costi di accesso (DLL, processi locali, processi remoti)

La applicazioni lavorano in modo **sincrono bloccante** in ogni **interazione applicativa** (chiamata locale o RPC) e possono richiedere operazioni delle interfacce **attraverso riferimenti non persistenti (**cioè **volatili** eventualmente da ricostruire**)**

- I clienti ritrovano i servizi in **componenti diversi** con diversi costi di supporto: in caso di processi sono **necessari proxy per ogni interfaccia** per i **diversi modi** (**proliferazione proxy**)
- I **proxy** devono essere preparati in **modo statico** ed essere **gestiti dal middleware** (per l'intero ciclo di vita)

Nuove specifiche con AUTOMATION

Sulla base della tecnologia **DCOM** la Microsoft ha sviluppato tutta la propria tecnologia di applicazione per il concentrato e il distribuito anche intervenendo nei *comitati di standard* e cercando di imporre le proprie scelte



DISTRIBUTED COM e AUTOMATION

Molte tecnologie laterali ma sulla base di

Automation: interfaccia universale (file ActiveX, OCX)

Con l'obiettivo di consentire ad un'applicazione (controller o client) **di comandare dall'esterno un'altra applicazione** (server) e di poterla **gestire da remoto**

Automation nasce dalla necessità di fornire un **metodo generale** per accedere a **COM** mediante **linguaggi dinamici di scripting** e, in generale, **linguaggi interpretati**

Visual Basic come meccanismo di interpretazione per ovviare alla necessità di creare un *wrapper diverso* per ogni interfaccia esportata da COM

*In pratica, si riescono a definire interfacce personalizzate e dinamiche **senza creare coppie proxy/stub** specifiche prima della esecuzione*

AUTOMATION

Tecnologia raccomandata in DCOM **Remote Automation**

Automation fornisce i **meccanismi** completamente **dinamici** per consentire ad un cliente (controller) **di "esplorare"** e **utilizzare a run-time un'interfaccia sconosciuta e permette di non avere la proliferazione dei proxy**

Automation fornisce un'interfaccia "auto-marshalled" standard universale alternativa alle *Vtables* e *IUnknown* per invocare i metodi esposti da un oggetto COM

una DLL di sistema *oleaut32.dll* funziona da **proxy/stub universale** per tutti i server di automation

In-place Activation

Attivazione differenziata dei gestori in modo statico o dinamico, *attivi sempre* o *solo se il sottocomponente è stato richiesto* (vedi OLE)

DCOM AUTOMATION

Structured storage

un intero file system all'interno di un singolo file

- Gestione di concorrenza e diritti di accesso
- Gestione delle transazioni

base per la futura gestione di file system ad oggetti (OFS)

Uniform data transfer

Unico sistema di scambio dati fra processi: clipboard, drag and drop, DDE *indipendentemente dal mezzo fisico usato per lo scambio (memoria, file, protocolli di rete ...)*

Lo scambio avviene mediante un oggetto COM detto Data Object

Il protocollo è costituito da una serie di funzioni API che hanno il compito di scambiare un Data Object fra due processi

AUTOMATION e WEB

Automation rappresenta la base di tutte le tecnologie Microsoft in ambito Intranet ed Internet (Web) e con il requisito della dinamicità

Active scripting

La possibilità di fare degli script che vadano oltre gli schemi statici (*html*)

Strumenti *client-side* (Internet Explorer)

Strumenti dalla parte del cliente, in genere compatibili solo con Microsoft browser

Strumenti *server-side* (IIS e ASP)

Strumenti dalla parte del server, in genere compatibili solo con Microsoft Internet Information Services

DINAMICITÀ: INTERFACCIA IDispatch

```
type IDispatch = interface(IUnknown)
  ['{00020400-0000-0000-c000-000000000046}']
HRESULT GetTypeInfoCount(out Integer Count);
HRESULT GetTypeInfo(Integer Index, LocalID;
     out TypeInfo * ptrInfoArea);
HRESULT GetIDsOfNames(const IID TGUID; Pointer Names;
     Integer NameCount, LocalID; out Pointer DispIDs);
HRESULT Invoke(Integer DispID; const IID TGUID;
     Integer LocalID; Word Flags; out Pointer Params;
     out Pointer VarResult, ExceptInfo, ArgErr);
end;
```

Interfaccia **IDispatch** eredita da **IUnknown** e permette di interrogare e conoscere la **interfaccia di un componente** durante la esecuzione **chiedendo al componente stesso (riflessione)**

⇨ **Componenti dinamici**

DETTAGLI: INTERFACCIA IDispatch

Per saperne di più della interfaccia IDispatch…

I metodi riflessivi preparano la Invoke

GetTypeInfoCount (out Integer Count) accerta la riflessione sul componente

GetTypeInfo (Integer Index, LocalID; out TypeInfo * ptrInfoArea);

consente di avere per un metodo la specifica dei tipi dei parametri per diversi ambienti

GetIDsOfNames (const IID TGUID; Pointer Names; Integer NameCount, LocalID;
out Pointer DispIDs);

permette di ottenere il displacement dei nomi dei parametri in formato interno per il metodo di interesse e con il contesto richiesto (come array in uscita, di nome metodo e tutti i parametri)

Invoke (Integer DispID; const IID TGUID; Integer LocalID; Word Flags;
out Pointer Params; out Pointer VarResult, ExceptInfo, ArgErr);

permette di specificare sia i parametri di ingresso / uscita, sia eventuali uscite corrette od anomale, richiedendo la invocazione del metodo stesso

INTERFACCIA IDispatch

L'insieme dei metodi accessibili **dinamicamente** tramite ***IDispatch*** prende il nome di *dispInterface*

GetTypeInfoCount *e* ***GetTypeInfo***
informazioni sui metodi esportati dall'interfaccia (*esplorazione*)

GetIDsOfNames *e* ***Invoke***
meccanismo di preparazione della chiamata dinamica
e dispatching (*utilizzo*)

Il meccanismo in estrema sintesi

IDispatch da solo permette di operare agevolmente su ogni oggetto sconosciuto, di cui è noto solo il nome …

ottenendo la conoscenza in modo dinamico interrogando l'oggetto stesso, che *risponde* ***in modo riflessivo*** su se stesso, poi abilitando la invocazione tramite automation

(vedi le classi per la *riflessione* e *introspezione* in Java)

INTERFACCIA IDispatch

Accesso all'oggetto

Passando il nome a COMOBJ si ricava il CLSID che si utilizza per ottenere l'interfaccia ***IUnknown*** e l'interfaccia ***IDispatch***

Questa viene usata per tutto il ciclo di richiesta

Per conoscere le proprietà:

`GetTypeInfo` restituisce l'interfaccia ***ITypeInfo*** che fornisce informazioni sui metodi della *dispinterface*: *nomi*, *elenco dei parametri* e *tipo dei valori di ritorno*

Per la Chiamata:

`GetIDsOfNames` con nome di metodo restituisce il suo ID e i parametri
`Invoke` invoca il metodo voluto in base all'ID e parametri e permette di conoscere il risultato

Params è un buffer di parametri (con tipo noto via GetTypeInfo)

VarResult contiene il valore restituito dal metodo invocato

COMPONENTI DCOM (AUTOMATION)

Componenti come entità capaci di fornire servizi in modo pervasivo
dovunque, per chiunque, in ogni momento
senza limiti alla riusabilità e **con inserimento dovunque**

Si noti che i componenti sono **più pervasivi** degli oggetti, a **grana più grossa** degli oggetti, ma con una **dipendenza più forte da un ambiente contenitore** che ne semplifica l'uso ma che è necessari per la loro esecuzione

VCL: legati agli ambienti Borland (Delphi, C++ Builder), (16/32 bit) - object oriented

JavaBeans: componenti di Java, Sun / Borland

VBX: (Microsoft) legati al Visual Basic (solo 16 bit) in disuso

OCX/ActiveX: basati su COM, indipendenti dal linguaggio (16/32 bit) - object-based

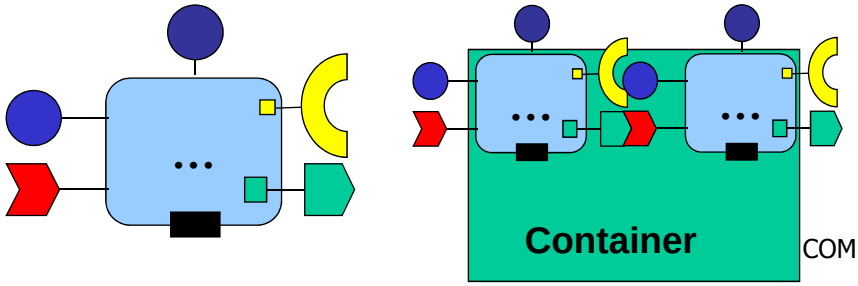
DCOM - COMPONENTI

Un **componente software** deve assomigliare a un **componente industriale** (vedi componente hardware)

Un componente deve avere

- **input** (come invocazioni richieste dall'esterno)
- **output** (come richieste da fare pervenire all'esterno)
- **stato** (come proprietà cui si possa accedere dall'esterno)

ed essere immerso in un ambiente di lavoro che semplifica il progetto (**container**) e che ha la *responsabilità della gestione dei componenti e ne semplifica il progetto*



DCOM - COMPONENTI PEM

Un **componente software** deve prevedere tre tipi di proprietà, spesso dette *piedini di interazione*:

proprietà, metodi, eventi o modello PEM

- **Proprietà** (property): "piedini di **stato**", pseudo-variabili che consentono di agire in modo protetto sullo stato interno
- **Eventi**: "piedini di **uscita**", in seguito a una occorrenza di qualcosa che si verifica nel componente, si vuole provocare l'esecuzione di metodi (**callback**) nel container
- **Metodi**: "piedini di **ingresso**", comandi che provocano l'esecuzione di azioni nel componente

Negli ambienti a componenti, l'applicazione in grado di incorporare componenti e di supportarne anche la interazione è il ***container***

COMPONENTI ACTIVEX/OCX

I **componenti ActiveX** basati su COM costituiscono in qualche modo il "riassunto" delle tecnologie basate su questo modello

- Usano la tecnica dei **documenti composti** (OLE Documents) per realizzare **l'in-place activation e la serializzazione**
- Usano Automation per **implementare proprietà, eventi e metodi** (**modello PEM**) e le capacità di introspezione

Sono implementati come **server in-process** (DLL)

- Inizialmente venivano chiamati OLE COntrols, poi OCX e infine ActiveX Components

Componenti come aggregati complessi da implementare, mai realizzati manualmente, ma attraverso strumenti a disposizione negli ambienti di sviluppo (Wizard)

IN-PLACE ACTIVATION

I componenti ActiveX sono **server OLE** in miniatura che implementano quasi tutte le interfacce OLE Documents

Diversamente dai normali server OLE (Excel dentro Word) che funzionano in modalità ***outside-in*** (**dinamici**)

- **Outside-in**: un oggetto Excel incorporato in un documento Word è inattivo normalmente e viene visualizzata una sua immagine statica (in formato metafile)
  Solo quando facciamo doppio click sull'immagine viene caricato Excel e l'oggetto diventa attivo

ActiveX funzionano in modalità ***inside-out*** (cioè **sempre pronti dal punto di vista della attivazione o statici**)

- ***Inside-out: i controlli ActiveX sono invece sempre "attivi" e la DLL rimane caricata per tutto il tempo di vita del form in cui il controllo è stato incorporato***
- ***Come se lavorassimo con aggregazione e sempre attivo il componente interno: non vediamo il ruolo del contenitore***

DCOM - COMPONENTI P.E.M.

Automation fornisce la tecnologia per implementare

sia i “piedini” del nostro **circuito integrato software**,

sia l’**introspezione** e la interazione con il **container**

L’**introspezione** realizzata dalle *type libraries* per un completo meccanismo di esplorazione delle capacità di un oggetto COM

- Le proprietà e i metodi vengono implementate sotto forma di un’interfaccia ***IDispatch*** del componente
- Le proprietà sono coppie di metodi per lettura (**get**) e scrittura (**set**) delle proprietà
- Il container mette a disposizione una interfaccia ***IDispatch*** per le variabili di ambiente

COMPONENTI - EVENTI

Implementazione degli eventi di Automation basata su

Il **componente** definisce nella sua type library l’elenco degli **eventi** che è in grado di innescare sotto forma di una dispatching interface (*dispInterface)* che non implementa

Il **container** legge dalla type library la definizione della *dispinterface* e provvede a fornire l’**implementazione**

A run-time, il componente richiede questa interfaccia e la utilizza ogniqualvolta si renda necessario innescare un evento

In DCOM si dice che il **componente** fa da ***source*** (sorgente) per la *dispInterface* (in pratica la definisce) e il **container** fa da ***sink*** (pozzo) (in pratica la implementa)

Il **container** non comunica direttamente con il componente ma attraverso **oggetti intermedi** definiti “connection point” e deve predisporre un altro oggetto intermedio detto “client site” per ogni controllo incorporato

DCOM ACTIVEX - SERIALIZZAZIONE

I **componenti** sono normalmente inseriti in un **container** che fornisce un contesto di attivazione e si incarica di tutte le funzioni di base

Il meccanismo di serializzazione deriva da OLE e si basa sulla tecnologia "structured storage"

> *Si utilizza lo stesso meccanismo che permette, ad esempio, il salvataggio dell'oggetto Excel nel documento Word*

Il componente salva o carica il proprio stato in questo storage
Il componente implementa l'interfaccia ***IPersistStorage*** che eredita da ***IStorage***

Il container chiama i metodi **Save** e **Load** passando un'interfaccia ***IStorage***

DCOM ACTIVEX - PERSISTENZA

```
IPersistStorage = interface(IPersist)
    ['{0000010A-0000-0000-C000-000000000046}']
 HRESULT IsDirty ();
 HRESULT InitNew(const IStorage stg);
 HRESULT Load(const IStorage stg);
 HRESULT Save(const IStorage stgSave;
                    BOOL fSameAsLoad);
 HRESULT  SaveCompleted(const IStorage stgNew);
 HRESULT HandsOffStorage ();
end;
```

Si possono salvare non solo *singoli componenti* ma anche *insiemi di container* e loro *stato complessivo* interno

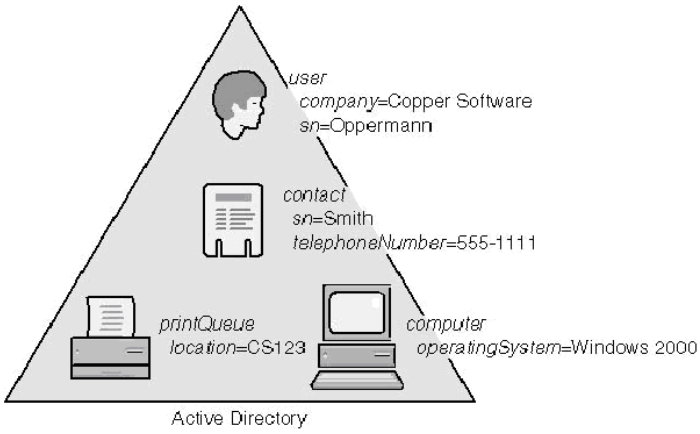
SISTEMA di NOMI GLOBALE

Active Directory usato sia in WinXX sia in .NET

Gli oggetti possono essere inseriti in un **directory** come X.500 per essere disponibili globalmente

Un oggetto in AD ha:

- **una classe** (descrive attributi)
- **uno schema** (per il directory)
- **uno spazio logico** (per essere trovato)
- **una organizzazione ad albero** (gerarchia di ricerca)
- **una organizzazione a foresta** (per aggregare alberi)



Uso di protocolli standard per l'accesso: LDAP

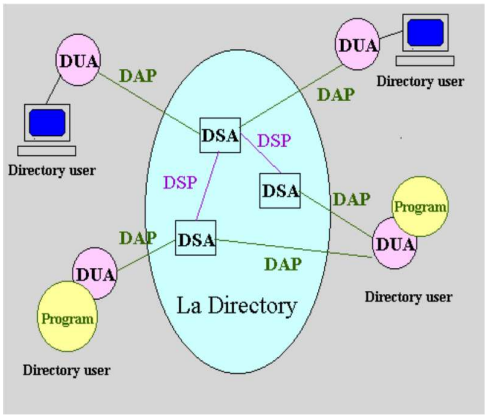
DIRECTORY X.500 e non DB

Directory memorizza informazioni su un **insieme di risorse** evitando duplicazioni di informazioni e problemi di sincronizzazione: **molte letture / poche scritture** consentendo

* **massima capacità espressiva**
* **gestione con molteplici autorità**
* **sicurezza partizionata e differenziata**

Al contrario di un db

* *attributi associati ai singoli oggetti*
* *oggetti indipendenti* tra di loro
* *relazione di contenimento* come base
* accesso *differenziato per singole entità*



Protocollo DAP per l'accesso standard LDAP per accesso da ambienti Internet

PROCESSI: modello dei THREAD

Il modello dei thread (da WIN32) per gestire processi pesanti

- Un **thread** esegue in un processo (**pesante e contenitore**) che viene automaticamente gestito dallo scheduler
- Ogni processo ha un thread principale con una **coda di messaggi** (**eventi prodotti** e **rilevati** dal sistema operativo)
- Ci sono funzioni per la gestione della sincronizzazione e comunicazione (**semafori**, regioni critiche, **eventi**, ecc.)
- **Non tutti componenti sono thread-safe (legacy)**

Apartment *come ambiente con una politica di threading fissata*

L'*appartamento* può essere:

***Single-threaded* STA** o ***Multiple-threaded* MTA (*Apartment*)**

- ***Un processo può avere più STA ma un solo MTA***
- C'è una sola coda di eventi per STA e una sola anche per un MTA e deve essere gestita dal *thread principale*

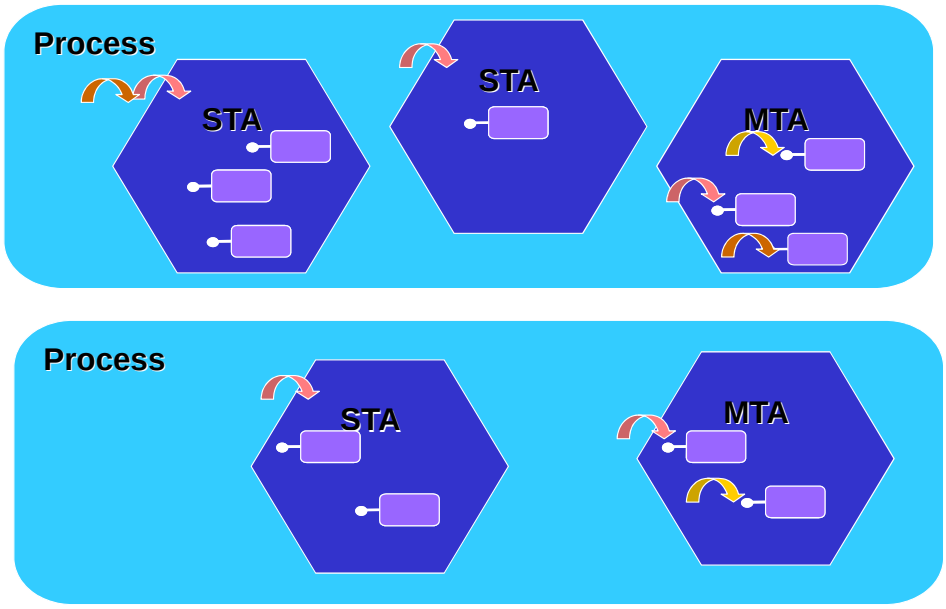
Un componente può essere destinato a STA solo, o anche MTA

APARTMENT e PROCESSI

Un processo contiene apartment (cioè più domini separati) con comunicazione via RPC

***Più Single-threaded* STA** e ***un solo Multiple-threaded* MTA**

Anche con thread del cliente che arrivano ed eseguono



SERVIZI ULTERIORI in DCOM

Molti servizi aggiuntivi

Supporto per **Garbage Collector Distribuito** (ping periodici)

Marshalling dei dati con politiche diverse

Interceptor sui canali ossia funzioni aggiuntive eseguite (codice) dalle due parti prima dell'azione sui dati (esempio: sicurezza)

Altri componenti correlati

Microsoft Transaction Server **(MTS)**

integrazione con transazioni

Microsoft Message Queue Server **(MSMQ)**

supporto messaggi

Microsoft Active Directory

versione di direttorio

COM+

ambiente a componenti

Microsoft Message Queue Server (MSMQ)

MSMQ esempio di MOM per il supporto di scambio di messaggi asincrono affidabile e persistente tra applicazioni distribuite

Un **Queue Manager** permette di **creare, distruggere, gestire le proprietà delle code**, e **gestire i messaggi** e gli accodamenti multipli considerando **qualità e servizio differenziato**

fare copie dei messaggi (log), un forte disaccoppiamento tra mittenti e destinatari

Ad esempio: messaggi con **priorità** (applicativa), servizi di **notifica**, **multicast**, **persistenza** (durata), supporto alla **sicurezza** (con requisiti diversi)

Il supporto ai messaggi richiede o un *database* o un *active directory* (a secondo della versione) per la persistenza e la qualità

Sono anche previsti connettori o esportazioni per altri ambienti

SUPPORTO COM+

COM+ come un **supporto** generalizzato a **componenti attraverso un supporto automatico di contenimento**

componenti COM+ devono eseguire in un container

COM+ non è vincolato a COM, ma compatibile con .NET

- supporto a proxy per la comunicazione remota con logica di **interceptor**
- **catalog** come repository degli attributi di un componente
- servizi supportati e forniti da COM+:
  - gestione eventi
  - supporto messaggi asincroni
  - supporto alle transazioni
  - supporto alla sicurezza
  - gestione pool di oggetti
- in .NET, uso di assembly

ALTRE FORME di COMUNICAZIONE

DCOM prevede un **supporto RPC standard sincrono per la comunicazione C/S**

la modalità standard può essere molto pesante e produrre ritardi eccessivi come unica modalità

Modi alternativi suggeriti

Chiamata all'indietro

⇨ **Interfaccia di callback**: il cliente provvede una **interfaccia di callback** per il servitore da chiamare al momento della risposta

Modo asincrono, via due metodi, prima richiesta e poi fine

⇨ **Metodo di fine**: si spezza la chiamata in due parti e il cliente provvede a recuperare il risultato al bisogno

ARCHITETTURA .NET e MIDDLEWARE

Il **passaggio** ad una **nuova architettura** sottostante (.NET 2002) - è una **piattaforma innovativa** a livello di macchina virtuale, più moderna e alternativa a JVM,) rappresenta una **forte discontinuità** nel **middleware**

Una opportunità di **ridefinire le modalità di comunicazione** ed **interazione** a livello di **middleware** (introducendo una **soluzione di continuità**, e con l'obiettivo di una **forte semplificazione**)

I sistemi **COM/DCOM** sono caratterizzati da **molteplici strumenti e strategie molto flessibili**, di basso livello, da personalizzare in molti modi, e a volte **difficili da utilizzare** (uso di aiuti e wrapper)

obiettivo: uso ottimizzato e flessibile

I sistemi **.NET** tendono a semplificare e a proporre **pochi semplici strumenti** per l'utente applicativo e modalità limitate e chiare (non troppo adattabili) per la **interazione**

obiettivo: uso semplificato

.NET

.NET piattaforma innovativa per arrivare ad un **framework multi-piattaforma** e **multi-linguaggio**

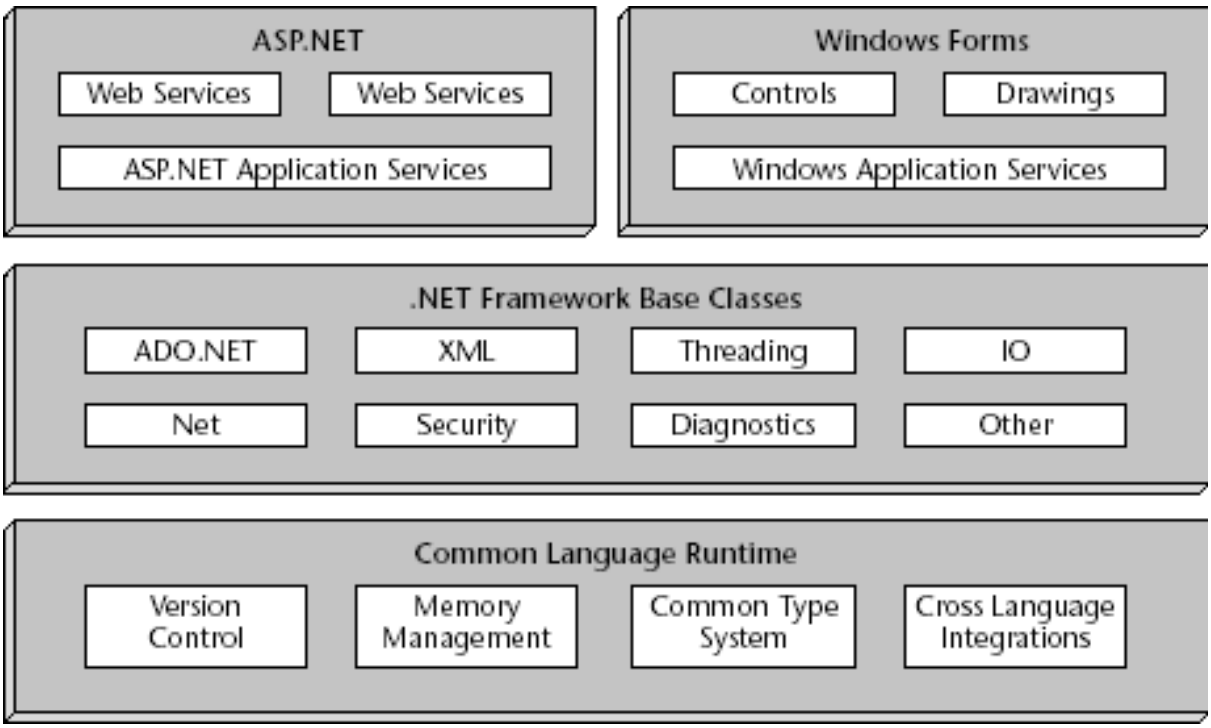
Macchina virtuale innovativa per la esecuzione di codice proveniente da qualunque linguaggio in modo nativo

CLR (Common Language Run-time)
parte di **macchina virtuale** di supporto indipendente dal linguaggio

Classi Base
le librerie delle classi che rappresentano i comportamenti elementari per la esecuzione

Strumenti e interfacce (ASP.NET e Windows forms)
strumenti per la interazione utente e per la gestione della parte di business (compatibilità Web)

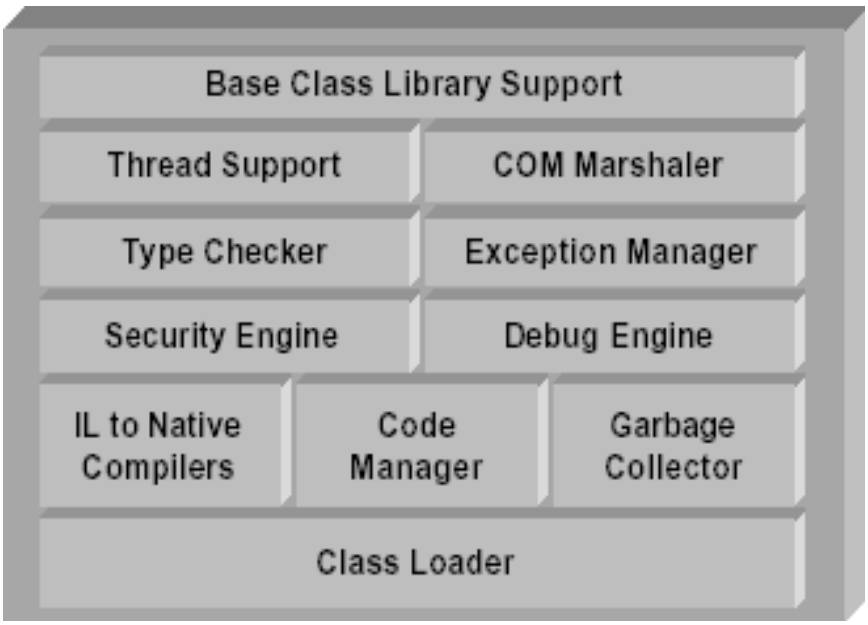
.NET



Common Language Runtime

CLR (Common Language Run-time) come supporto

CLR carica, supporta, gestisce il codice in formato intermedio



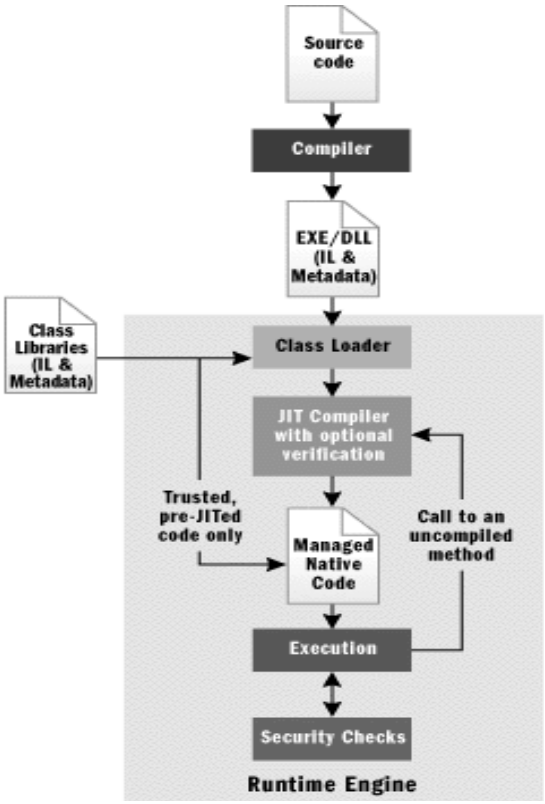
CLR gestisce:
versioning,
memoria,
integrazione
verso la architettura locale,
…

Intermediate Language e Metadati

CLR prevede **portabilità per eseguibili** che possano anche migrare usando le informazioni di **metadati** che ne descrivono la organizzazione ed i componenti

Forte vocazione alla compatibilità
I formati compilati sono disponibili solo per la architettura target

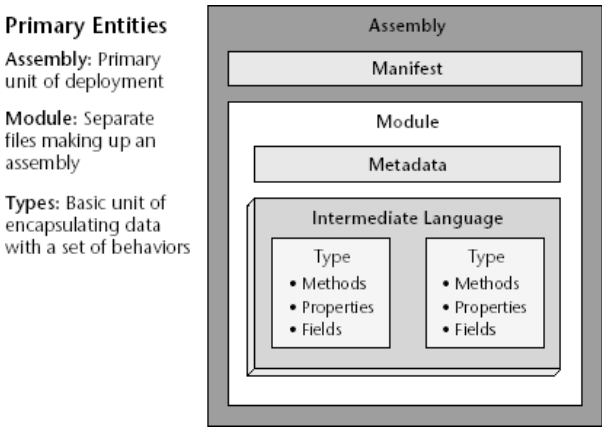
Class Loader
JIT (Just In Time) compiler



Assembly come Metadati

Assembly descrive **informazioni** sulla applicazione e per la sua corretta esecuzione

- identità
- lista file
- altri assembly riferiti
- lista risorse interne ed esportate
- permessi necessari



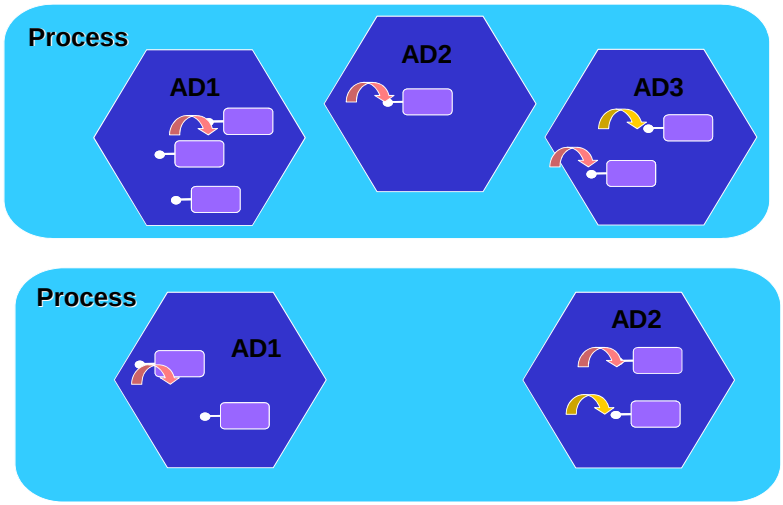
Ogni assembly è caricato in un processo separato
ma è possibile richiedere **assembly condivisi** (con nome unico)

APPLICATION DOMAIN

Application Domain

nell'ambito di uno **stesso processo** possiamo avere ambiti separati e minimizzando la interferenza reciproca

Il CLR controlla che non si passi da uno all'altro direttamente, ma solo in modo protetto, come tra processi diversi



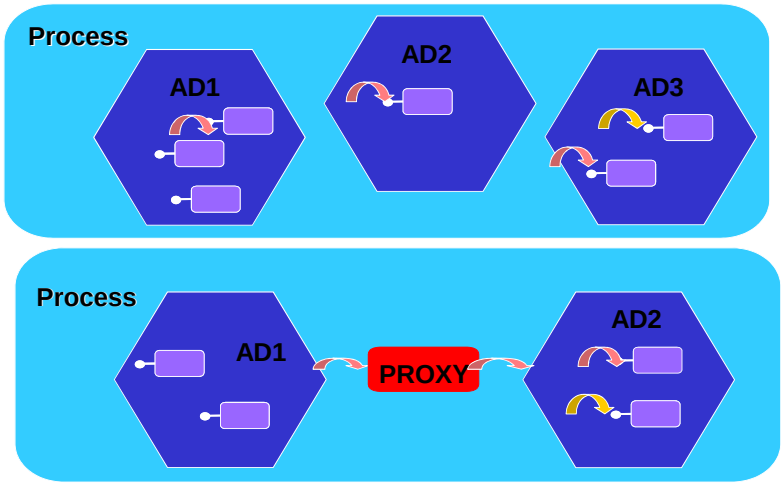
Necessità di **intermediari** che separano e proteggono gli **ambienti mantenuti distinti**

COMUNICAZIONE REMOTA

Uso di meccanismi diversi per la comunicazione tra domini diversi:

.NET remoting, Web Services, DCOM

In genere, si usano meccanismi differenziati e disponibili a scelta dell'utilizzatore



sono previsti oggetti **proxy** che separano e proteggono gli **AD**

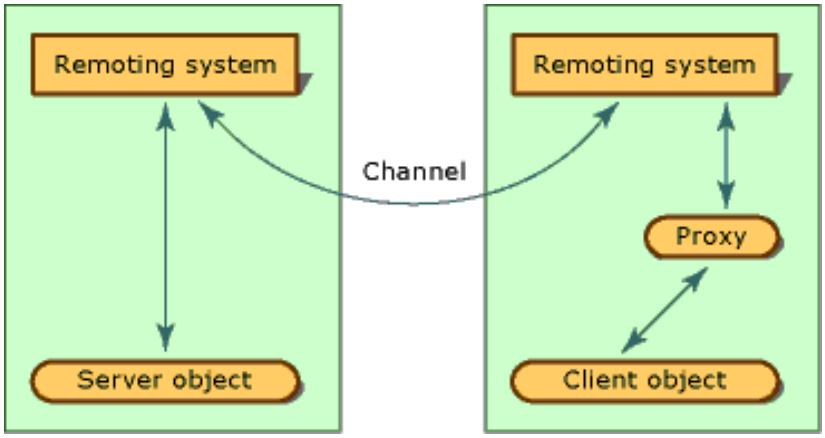
.NET REMOTING

.NET remoting

meccanismi per la **comunicazione sincrona** tra oggetti remoti

Un **proxy** viene creato dal supporto per il cliente che vuole riferire (attivare) un **oggetto remoto** (visibile)

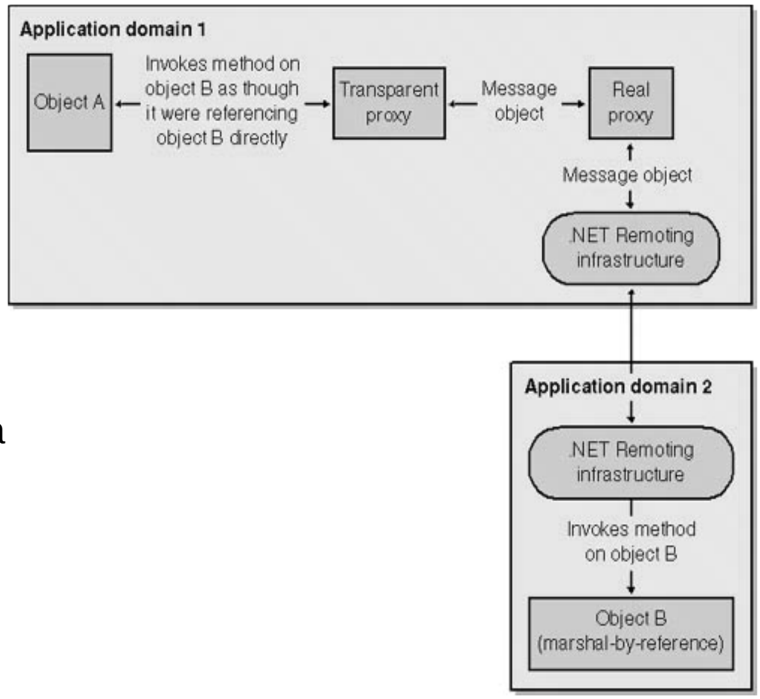
Il **proxy** tiene conto anche delle possibili eterogeneità tra sistemi diversi



Oggetti PROXY

PROXY

Insieme di **meccanismi** per il passaggio di informazioni e di controllo tra ambiti diversi



Diversi proxy…

Proxy trasparente con interfaccia uguale all'oggetto remoto creato e che intercetta il supporto .NET

Proxy reale come mezzo di passaggio da un dominio ad un altro in modo dinamico via stack intermedio

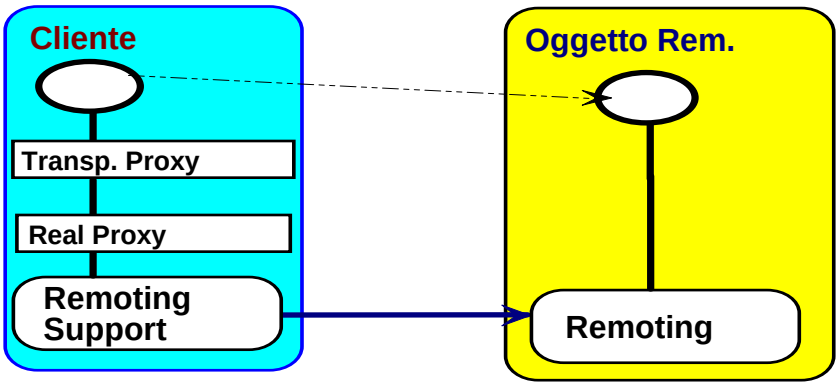
Oggetti Reference via PROXY CHAIN

Object Reference

Un oggetto **visibile da remoto** viene registrato all'application domain del nodo di creazione e permette di rendere visibile l'oggetto da remoto

Il **proxy trasparente** **è immutabile** e solo copiabile sui nodi interessati

Il **proxy reale** potrebbe anche essere modificato e svolgere funzioni dalla parte del cliente per il bilanciamento di carico



Oggetti CHANNEL

Canali

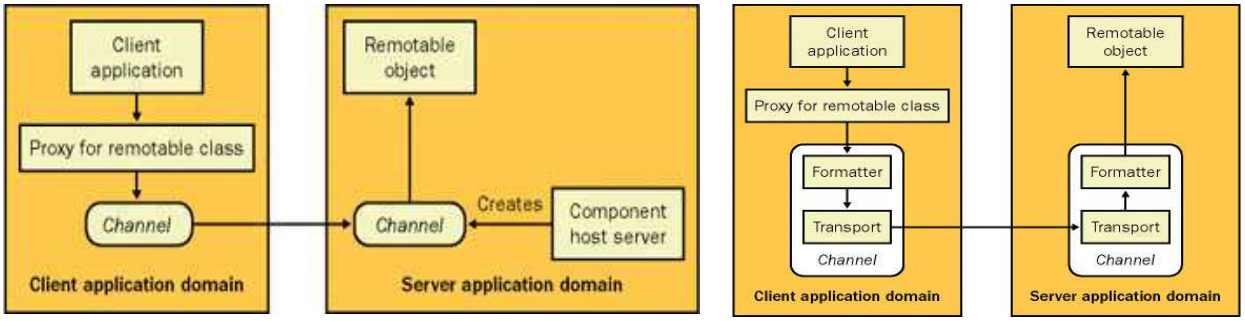
oggetti di supporto alla comunicazione tra **application domain** diversi

per supportare il passaggio dei parametri e consentire un corretto trasporto delle informazioni

Gli **application domain** registrano i **canali** che i clienti possono utilizzare e tra cui possono scegliere il veicolo della comunicazione

I canali sono **spesso bidirezionali** e possono consentire anche di dare accesso ad oggetti **remoti diversi**

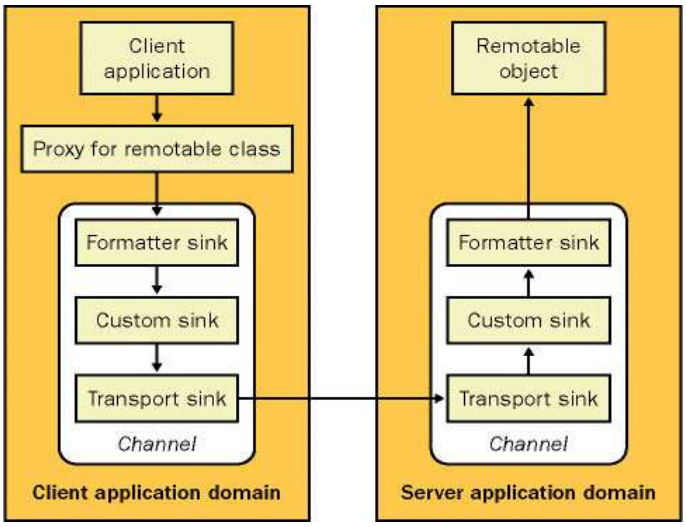
I **due** tipi di **canali standard** prevedono solo i protocolli **TCP** **e** **HTTP**



CHAIN nei CANALI

Catene di responsabilità

I canali gestiscono e controllano il messaggio attraverso catene di oggetti detti **sink** che possono dividersi i compiti e la responsabilità di gestione



Ad esempio:

Formatter sink

per la presentazione dei dati

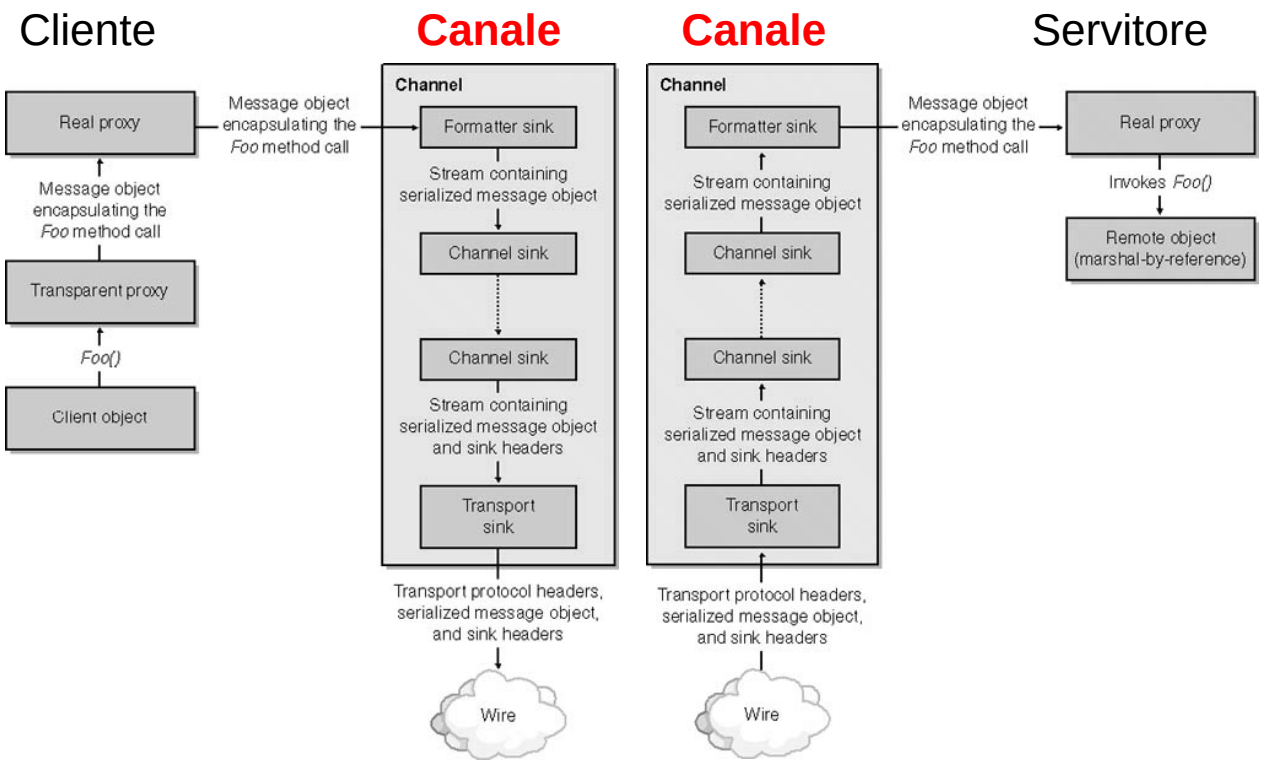
Custom sink

per azioni application-dependent

Transport sink

per la comunicazione vera e propria

VISIONE di INSIEME

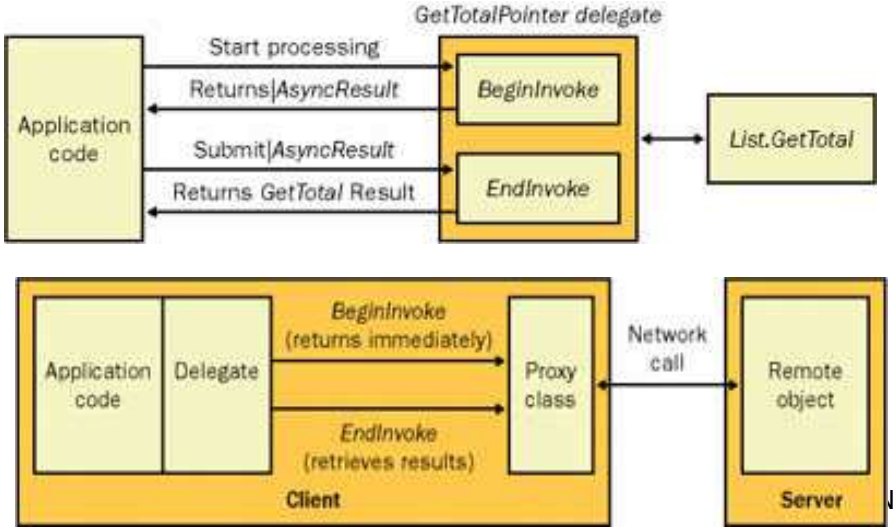


OPERAZIONI REMOTE ASINCRONE

Operazioni remote sono sincrone a default

Ma sono possibili operazioni asincrone via Callback

operazioni asincrone attraverso **callback** (come puntatori a funzioni delegate ad essere eseguite) e **eventi** (di basso livello nascosti)



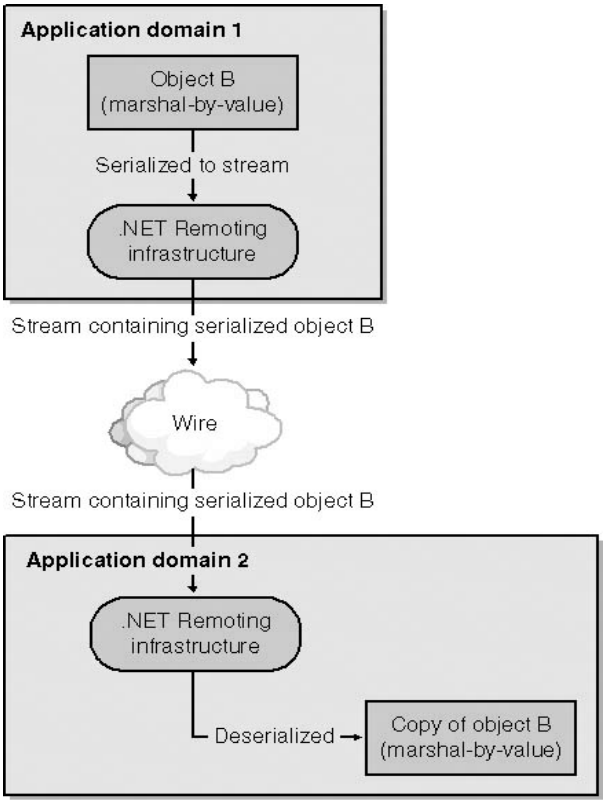
PASSAGGIO ARGOMENTI: VALORE …

In caso di **operazione remota**, si prevedono passaggi **per valore** e per **riferimento**

In caso di **passaggio per valore, l'oggetto viene copiato** sul **nodo remoto**

La serializzazione deve essere gestita in modo automatico dal supporto

Problema: **overhead**



ARGOMENTI per RIFERIMENTO

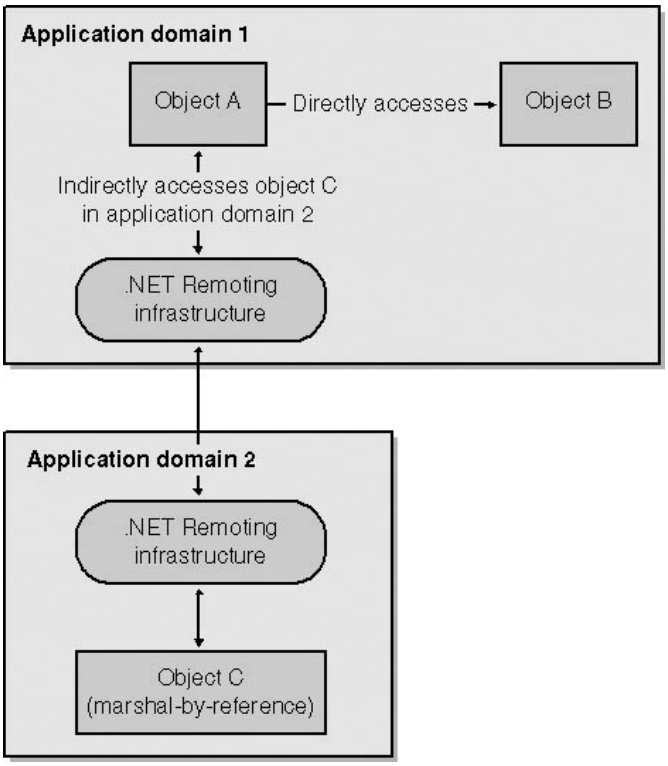
In caso di passaggio per **riferimento,**

Si crea un **riferimento remoto per l'oggetto** che possa essere raggiunto **dai nodi remoti**

Nel caso si acceda da altri application domain, si passa un **proxy**, creato al momento, per riferire l'oggetto remoto

Il proxy viene utilizzato solo nel caso che si acceda da remoto

Non necessario se si lavora nello stesso application domain



CICLO di VITA degli OGGETTI REMOTI

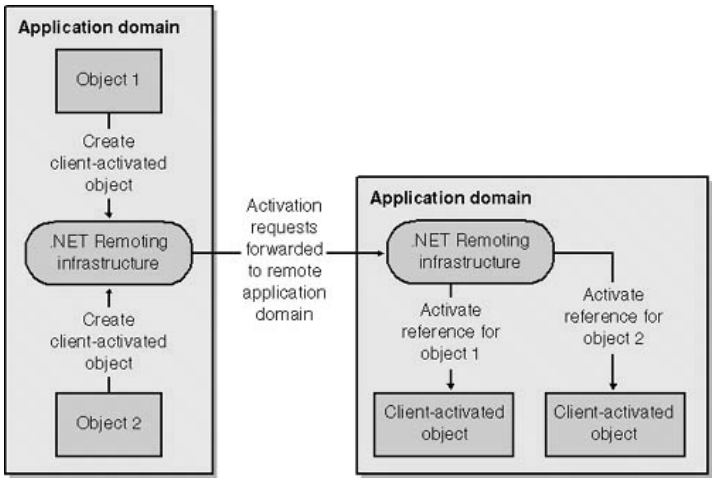
In caso di oggetti remoti**,**

una possibilità è che il controllo sia tutto sotto la responsabilità del cliente

client-activated

*L'oggetto rimane in memoria finché il cliente è **attivo** ed ha un riferimento all'oggetto stesso*

Mantenimento dello stato



Il cliente specifica un **lease**, durata attesa della vita, oltre la quale l'oggetto stesso viene deallocato e distrutto a meno che il lease non sia confermato

Problema: **gestione dei lease**

Il server può anche contattare il cliente per fare un verifica prima della deallocazione

OGGETTI REMOTI SERVER-ACTIVATED

gli **oggetti remoti** **possono essere attivati e gestiti dal server secondo le sue politiche interne (o server-activated)**

L'oggetto rimane in memoria secondo le politiche del server

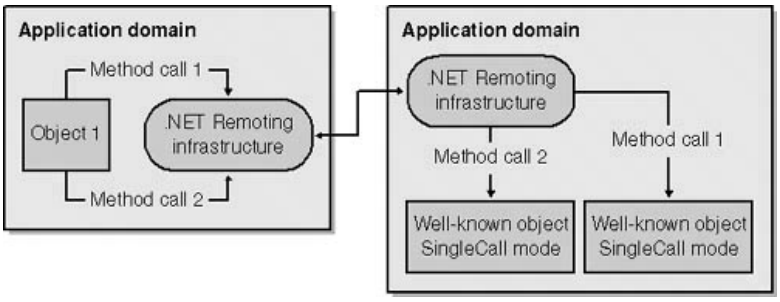
Due politiche principali:

- **singleton**
- **single-call**

Distinte per le diverse strategie di allocazione e per la im/possibilità di avere stato

Single-call

Ogni invocazione produce attivazione per la sola durata della operazione richiesta



OGGETTI REMOTI SERVER-ACTIVATED

Singleton

La prima invocazione produce l'attivazione per la operazione richiesta

L'oggetto sopravvive per la esecuzione anche di altre operazioni

Lo stato non è garantito perché è sotto il controllo del supporto che può decidere deallocazione e riallocazione a piacere