Il modello COM

Enrico Lodolo
e.lodolo@bo.nettuno.it

Integrare le applicazioni: da clipboard a OLE

- **Windows è nato con un obiettivo ben preciso: spostare l'integrazione fra funzioni applicative a livello di ambiente operativo: l'utente può quindi scegliere sul mercato gli applicativi specializzati migliori lasciando al sistema operativo il compito di integrarli.**
- **Per realizzare questo obiettivo deve mettere a disposizione meccanismi efficaci per la comunicazione fra applicazioni**
- **L'evoluzione del sistema è strettamente legata all'introduzione di meccanismi sempre più potenti per consentie questa comunicazione**
- **Si è passati dalla clipboard, al DDE (Dynamic data exchange, un protocollo di comunicazione asincrono basato sui messaggi) a OLE**
- **OLE consente di inserire all'interno di un documento prodotto da un'applicazione (p.es. Word) un sottodocumento prodotto e gestito da un'altra applicazione (p.es una tabella Excel)**
- **La prima versione di OLE, risalente al '90, si basava su DDE ed era molto fragile e lenta**
- **Era necessario trovare un meccanismo più robusto ed efficiente su cui poggiare OLE**

Problemi del modello Windows

- **Il modello di base dei sistemi windows è cresciuto molto a partire dalla versione 1.0 (1985)**
- **Questa crescita è avvenuta senza sostanziali adeguamenti di architettura**
- **L' API è molto vasta (più di 1000 funzioni), cresciuta disordinatamente, con nomi di funzioni attribuiti senza una regola precisa**
- **Le DLL costituiscono una buona base per la modularità ma presentano alcuni inconvenienti:**
  - Dipendenza dalla collocazione fisica: le applicazioni caricano le DLL facendo riferimento al path in cui si trovano e se una DLL viene spostata le applicazioni non riescono più ad accedere ai servizi
  - Gestione delle versioni: le DLL non possiedono meccanismi intrinseci di gestione delle versioni. Questa problematica viene lasciata alla buona volontà e alla disciplina degli svilluppatori con grossi rischi di compatibilità.
- **Occorre un meccanismo migliore per comunicare fra le applicazioni e sistema operativo e in generale fra fruitori e fornitori di servizi**

La storia si ripete

- **Le singole applicazioni sono cresciute a dismisura mantenendo una struttura essenzialmente monolitica.**
- **La continua aggiunta di funzionalità le fa assomigliare sempre più ai pacchetti integrati della prima metà degli anni ‘80.**
- **Si hanno notevoli sovrapposizioni e duplicazioni di funzioni fra word processors, fogli elettonici, database ecc.**
- **Nasce quindi la necessità di scomporre questi applicativi “monstre” in moduli che possano essere condivisi**
- **E’ necessario procedere ad una “componentizzazione” delle applicazioni e ancora una volta deve essere il sistema operativo a fornire gli strumenti**
- **In tal modo un utente potrà scegliere di installare solo le funzionalità che gli sono necessarie attingendo ad un mercato di componenti sostituibili fra di loro**

Un abbozzo di soluzione..

- **Una tecnologia basata su oggetti e costruita su un modello di interazione client-server di tipo sincrono rappresenta una soluzione ideale per questo tipo di problemi**
- **Infatti:**
  - **Robustezza:** un meccanismo di interazione sincrono permette di costruire una forma di comunicazione intrinsecamente robusta
  - **Omogeneita dell'API:** un'impostazione object-based prevede la suddivisione dei servizi forniti dal sistema operativo in interfacce - (interfaccia=insieme dei metodi di un oggetto) e quindi fornisce una classificazione ordinata e coerente dei servizi stessi (cfr. Java)
  - **Indipendenza dalla collocazione:** un'interazione di tipo client/server si basa solo su un protocollo di comunicazione e su un meccanismo di indirizzamento: è quindi totalmente indipendente dalla collocazione fisica dei soggetti e dalla loro implementazione
  - **Componentizzazione:** il principio di incapsulamento è il cardine di qualunque tecnologia di componenti software

COM

- **COM = Component Object Model (1992)**
- **E'un modello binario di interazione fra processi, basato su componenti e su un meccanismo di comunicazione client/server di tipo sincrono.**
- **COM è un modello, cioè un insieme di specifiche, supportato da alcuni servizi di sistema (supporto runtime)**
- **Le specifiche definiscono uno standard binario per la creazione di componenti in grado di interagire fra di loro**
- **Trattandosi di uno standard binario c'è completa indipendenza dal linguaggio di programmazione usato per realizzare i componenti e le applicazioni che li utilizzano**
- **Il modello prevede sia un funzionamento locale che distribuito (DCOM)**
- **Il supporto runtime è costituito da una DLL di sistema che fornisce i servizi necessari per l'accesso ai componenti**

Indirizzamento e GUID

- COM deve prevedere un meccanismo di indirizzamento per reperire i componenti
- L'indipendenza dalla collocazione fisica non consente di utilizzare un indirizzo fisico (pathname)
- Si utilizzano invece degli identificatori globali (GUID=globally unique identifiers)
- Il concetto di GUID è stato introdotto, con un nome leggermente diverso (UUID=universally unique id.), dall'OSF (Open Software foundation) nelle specifiche DCE (Distributed computing environment).
- In DCE gli UUID vengono utilizzati per identificare i destinatari delle chiamate di procedura remota (RPC)
- Un GUID è un numero di 128 (16 byte) bit assegnato in modo da garantire l'unicità nello spazio (48 bit) e nel tempo (60 bit).
- Viene rappresentato così: {32bb8320-b41b-11cf-a6bb-0080c7b2d682}
- COM utilizza diversi tipi di GUID

CLSID

- **Il primo utilizzo dei GUID è nell'identificazione delle classi di componenti: ogni classe di componenti COM è caratterizzata da un proprio identificatore che viene chiamato CLSID (Class Identifier)**
- **Disponendo di un CLSID si può chiedere a COMPOBJ di creare un istanza e restituire un riferimento**
- **Il database di sistema di Windows - la registry - mantiene una corrispondenza fra CLSID e le entità fisiche (EXE, DLL) che contengono l'implementazione dei componenti (server)**
- **La funzione CoCreateInstance (contenuta in COMPOBJ) provvede a:**
  - reperire il server tramite la registry
  - caricarlo in memoria (se non è già presente)
  - chiamare una funzione che crea un'istanza e restituisce un riferimento
- **La registry fornisce anche un servizio di naming, ovvero una corrispondenza fra nomi di classi (ProgID) e CLSID**
- **I ProgID hanno sono stringhe con il formato <nome>.<componente>.<versione>**
  - P.es. Word.WordBasic.5

Comunicare con i componenti: le interfacce

- **Per definizione un oggetto COM è un oggetto identificato univocamente da un GUID in grado di esporre una o più interfacce**
- **Un interfaccia è insieme di funzioni (metodi) che permettono di interagire con l'oggetto che la espone**
- **In virtù dell'incapsulamento, le interfacce sono l'unico modo per interagire con l'oggetto**
- **COM è stato pensato in C++ e il meccanismo delle interfacce nasce dalla tecnica abitualmente usata per accedere ad oggetti definiti e istanziati nelle DLL.**
- **Un'interfaccia non è altro che un puntatore ad una porzione della "virtual method table" (*vtable o VMT*) di un oggetto**
- **La VMT è in sostanza una tabella di puntatori a funzione**
- **Usando una terminologia "Java" un oggetto COM è un oggetto che implementa un certo numero di interfacce**
- **Anche le interfacce sono identificate da GUID. I GUID che svologono questo ruolo vengono chiamati IID (Interface Identifiers)**
- **Per convenzione i nomi delle interfacce iniziano con la lettera I**

IUnknown: il punto di partenza

- **Un Oggetto COM deve esporre almeno una interfaccia: IUnknown**
- **IUnknown rappresenta una sorta di punto di ingresso in quanto consente di accedere a tutte le altre interfacce esposte dall'oggetto**
- **In pratica un oggetto COM si identifica con la sua IUnknown**
- **IUnknown comprende 3 metodi: QueryInterface, AddRef e Release**
- **AddRef e Release implementano un meccanismo di "reference counting"**
- **QueryInterface permette di accedere alle altre interfacce esposte dall'oggetto: possiamo chiedere all'oggetto se implementa una determinanta interfaccia passando come parametro il relativo IID. In caso positivo il parametro Obj contiene un riferimento all'interfaccia richiesta**
- **La sintassi è:**

```
function QueryInterface(const IID:TGUID; out Obj):HRESULT;
```

- **HRESULT è un intero che ci dice se l'interfaccia richiesta è disponibile (S_OK) oppure no (E_NOINTERFACE)**
- **Anche IUnknown è identificata da un IID:**

```
'{00000000-0000-0000-C000-000000000046}'
```

IUnknown: la madre di tutte le interfacce

- **COM supporta l'ereditarietà delle interfacce, quindi un meccanismo per il polimorfismo ma non per il riuso**
- **Tutte le interfacce COM discendono da IUnknown**
- **Questo significa che tutte le VMT corrispondenti hanno nei primi 3 slot i metodi QueryInterface, AddRef e Release**
- **La presenza di QueryInterface consente di passare da un'interfaccia all'altra senza dover ritornare sempre indietro ad IUnknown**
- **L'ereditarietà delle interfacce è usata molto limitatamente: in pratica tutte le interfacce discendono da IUnknown o da IDispatch (che discende a sua volta da IUnknown)**

Reference counting

- Come abbiamo detto gli oggetti COM implementano un meccanismo di reference counting per le interfacce
- In pratica ogni interfaccia ha un suo contatore
- Quando QueryInterface restituisce un'interfaccia incrementa anche il contatore
- Ogni chiamata ad AddRef incrementa a sua volta il contatore
- Ogni chiamata a Release lo decrementa e quando il conteggio scende a zero l'interfaccia può essere liberata
- Quando i contatori di tutte le interfacce implementate da un oggetto sono a zero l'oggetto può essere distrutto
- E' una sorta di "garbage collection manuale"
- In pratica quando l'applicazione che utilizza l'oggetto (il client) assegna il riferimento ad una nuova variabile deve provvedere a chiamare AddRef. Inoltre deve chiamare Release ogni volta che alla variabile viene assegnato un nuovo riferimento oppure quando la variabile stessa esce dallo scope

Schema di utilizzo di un oggetto COM

- **Vediamo in pratica come funzionano le cose per un'applicazione che usa un'oggetto COM**
- **Inizializza il sistema chiamando CoInitialize**
- **Chiama la funzione CoCreateInstance, esportata da COMPOBJ.DLL, passando come parametro il CLSID dell'oggetto che ci interessa**

```
function CoCreateInstance(const clsid:TCLSID;unkOuter:IUnknown;
            dwClsContext:Longint;const iid:TIID;out pv): HResult;
```

- **CoCreateInstance procede così:**
  - usa la registry per risalire al server che implementa la classe richiesta
  - se la classe è registrata attiva il server (se non è gia attivo)
  - chiede al server di creare un istanza
  - riceve dal server un riferimento all'interfaccia IUnknown dell'istanza
  - restituisce unknown restituisce all'applicazione
- **L'applicazione usa IUnknown.QueryInterface per accedere all'interfaccia voluta**
- **Può invocare tutti i metodi disponibili e accedere ad altre interfacce**
- **Usa AddRef e Release per gestire il tempo di vita dell'oggetto**
- **Alla fine di tutto chiama CoUninitialize**

Creazione degli oggetti: Class Factory

- **Consideriamo il caso di un server implementato in una DLL**
- **La DLL esporta due funzioni per registrare e deregistrare il server nella registry.**
- **Si usa un utility di sistema (regsvr32) per eseguire la registrazione**
- **La DLL esporta una funzione per la creazione degli oggetti:**
  ```
  function DllGetClassObject(const CLSID,IID:TGUID;var Obj):HResult;
  ```
- **Il meccanismo di creazione è indiretto: la funzione non crea un'istanza della classe richiesta**
- **Crea invece un'istanza di una "Class Factory" e ne restituisce l'interfaccia IClassFactory**
- **Chiamando il metodo IClassFactory.CreateInstance si ottiene finalmente la creazione dell'istanza e la restituzione di IUnknown**
- **Il meccanismo delle Class Factory consente di incapsulare le problematiche relative alla creazione di un oggetto e di mantenere un elevato grado di isolamento fra client e server**
- **CoCreateInstance chiama prima la funzione CoGetClassObject per ottenere la class factory e quindi invoca IClassFactory.CreateInstance**

Ereditarietà e aggregazione

- **Come abbiamo detto, COM supporta l'ereditarietà delle interfacce ma non quella delle implementazioni.**
- **Nella visione dei progettisti di COM l'ereditarietà provoca un'accopiia,mento troppo stretto fra gli oggetti**
- **In particolare l'ereditarietà è vista come una pericolosa breccia nell'incapsulamento: una classe derivata può accedere allo stato interno della classe base**
- **Questo può provocare interazioni non desiderate e creare problemi di compatibilità fra versioni**
- **In alternativa COM propone un meccanismo di riuso chiamato aggregazione (una forma di delega): un oggetto espone anche le interfacce di un altro oggetto (aggregato)**
- **E' molto meno elegante e più complessa dell'ereditarietà (poco più di una specifica di implementazione)**