COM
Automation

Enrico Lodolo
e.lodolo@bo.nettuno.it

Automation: un'interfaccia universale

- **Automation nasce dalla necessità di fornire un metodo generale per accedere a COM mediante linguaggi di scripting e linguaggi interpretati in generale.**
- **E' stata inventata dai progettisti di Visual Basic che si erano trovati di fronte alla necessità di creare un wrapper diverso per ogni interfaccia esportata da COM**
- **E' un'interfaccia standard ma universale, che fornisce un metodo alternativo alle *vtables* per invocare i metodi esposti da un oggetto COM**
- **In pratica, seppur con qualche limitazione, si riesce a definire interfacce personalizzate senza ricorrere a MIDL per creare coppie proxy/stub specializzate**
- **Un'interfaccia basata su Automation è "auto-marshaled", in quanto la DLL di sistema *oleaut32.dll* funziona da proxy/stub universale per tutti gli automation server**
- **E' un meccanismo completamente dinamico: consente ad un client (controller) di "esplorare" e utilizzare a runtime un'interfaccia sconosciuta**

IDispatch e dispinterfaces

- **Alla base di tutto c'è l'interfaccia IDispatch:**

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

- **Come tutte le interfacce anche IDispatch discende da IUnknown**
- **L'insieme dei metodi accessibili dinamicamente tramite IDispatch prende il nome di *dispinterface***
- **Nella dichiarazione abbiamo due gruppi di metodi:**
  - GetTypeInfoCount e GetTypeInfo forniscono informazioni sui metodi esportati dall'interfaccia (esplorazione)
  - GetIDsOfNames e Invoke realizzano invece il meccanismo di chiamata dinamica o dispatching (utilizzo)

Il meccanismo in sintesi

- **Grazie a IDispatch un interprete è in grado di operare agevolmente su un oggetto sconosciuto, di cui è noto solo il nome.**
- **Accesso all'oggetto:**
  - Passando il nome a COMPOBJ si ricava il CLSID
  - Utilizzando il CLSID si ottiene l'interfaccia IUnknown
  - Con IUnknown.QueryInterface si ottiene l'interfaccia IDispatch
- **Esplorazione:**
  - IDispatch.GetTypeInfo restituisce l'interfaccia ITypeInfo che fornisce informazioni sui metodi della dispinterface: nomi, elenco dei parametri e tipo dei valori di ritorno.
- **Chiamata:**
  - IDispatch.GetIDsOfNames prende il nome di un metodo e restituisce il suo ID (DispID)
  - IDispatch.Invoke è in grado di invocare il metodo voluto in base all'ID
  - Params è un buffer che contiene i parametri (il cui tipo è noto grazie a GetTypeInfo)
  - VarResult contiene il valore restituito dal metodo invocato

Client-side marshaling e late binding

- **In pratica IDispatch lascia al al controller il compito di eseguire esplicitamente il marshaling:**
  - GetTypeInfo fornisce le informazioni per comporre il PDU
  - I parametri passati a IDispatch.Invoke (DispID, Params, VarResult ...) sono i vari pezzi del PDU
- **L'unica cosa lasciata alla coppia proxy/stub è la trasmissione sul canale**
- **In pratica in genere è l'interprete (Visual Basic o VBScript ...) o il compilatore (Delphi, VB 5.0) ad eseguire dietro le quinte queste operazioni**
- **Nel nostro programma troveremo semplicemente un'istruzione di questo tipo:**

```
AInteger:=MyAutoObject.DoSomething(ADouble,AString);
```

- **Questa tecnica di chiamata tramite GetIDsOfNames/Invoke viene definita "late binding" in quanto viene risolta a tempo di esecuzione.**
- **Il compilatore non è in grado di rilevare se la chiamata non esiste o i parametri sono errati: la gestione degli errori viene fatta a runtime.**

Limiti e prestazioni

- **L'unica limitazione effettiva è costituita dal numero limitato di tipi per i parametri e i valori di ritorno (in pratica i tipi riconosciuti da GetTypeInfo) ma non si tratta di una restrizione particolarmente severa**
- **Non è possibile passare strutture ma si possono passare interfacce che incapsulano strutture e inoltre esistono dei tipi "stream" che permettono di passare grosse quantità di dati in una sola volta**
- **La suddivisione della chiamata fra GetIDsOfNames (eseguita una volta per tutte, eventualmente passando più nomi di metodi) e Invoke (chiamata più volte) rappresenta una forma di ottimizzazione**
- **Una chiamata di questo tipo è 10 volte più lenta di una chiamata mediante *vtable* se il server è in-process (DLL). Nel caso di local server il rapporto scende a 2 e si ha una sostanziale equivalenza nel caso di server remoti (DCOM)**
- **Si può avere un'ulteriore ottimizzazione mediante il cosiddetto "ID binding": il compilatore chiama GetIDsOfNames memorizzando gli ID nel codice generato e quindi a runtime si usa solo Invoke**

Type Libraries

- In pratica GetTypeInfo e l'interfaccia ITypeInfo sono poco utilizzate
- Sono molto più utilizzate le *type libraries* che consentono di ottenere in un colpo solo tutte le informazioni su un server
- Una *type library* è una struttura binaria, che viene inserita nel server sotto forma di risorsa (come un icona o un bitmap), e che può quindi essere letta agevolmente da un interprete o da un compilatore
- Originariamente venivano scritte in forma sorgente con una sintassi descrittiva simile a quella di IDL denominata ODL e poi trasformate in file binari (.TLB) da un opportuno compilatore
- Con l'uscita di MIDL, ODL è stato incluso in quest'ultimo e quindi si utilizza lo stesso compilatore
- Come per le altre risorse si va diffondendo l'uso di editor che generano direttamente i file .TLB e le definizioni da includere nei programmi (.h per C/C++ o unit apposite nel caso di Delphi)
- Le type libraries contengono anche gli ID dei metodi e quindi GetIDsOf Names non viene più utilizzato.
- In pratica si utilizza solo IDispatch.Invoke

Dual interfaces

- **Un'evoluzione interessante di Automation (anche in termini di prestazioni) è costituita dalle cosiddette *dual intrerfaces***
- **In generale i metodi resi accessibili mediante *dispinterfaces* sono metodi interni dell'oggetto e si fa comunemente uso di una tabella di puntatori per far corrispondere ID e funzioni**
- **Questa struttura è molto simile ad una *vtable* e viene abbastanza naturale pensare di esporre anche questi metodi nell'interfaccia dell'oggetto**
- **In pratica una *dual interface* è semplicemente un'interfaccia che eredita da IDispatch e che comprende tutti i metodi accessibili attraverso Invoke**
- **In tal modo si può scegliere se invocare i metodi direttamente (molto comodo per i compilatori se l'interfaccia è nota a tempo di compilazione) oppure via *dispinterface* (va bene per gli interpreti o quando si vuole accedere agli oggetti in modo completamente dinamico)**
- **La chiamata diretta è ovviamente più efficiente e si parla in questo caso di *early binding* in quanto è risolta a tempo di compilazione**

Dual interfaces e marshaling automatico

- **Anche se si usano le chiamate dirette non è necessario generare la coppia stub/proxy con MIDL**
- **Infatti OLEAUT32.DLL è in grado comunque di gestire il marshaling automaticamente, grazie alle informazioni contenute nella *type library,* e a fungere quindi da proxy/stub universale**
- **E' una tecnica molto comoda ed è infatti la più utilizzata, soprattutto nei casi di elaborazione distribuita (DCOM)**
- **Eliminati i problemi di prestazioni l'unica limitazione che rimane è quella su tipi di dati utilizzabili ma, come abbiamo detto, nella maggior parte dei casi non sostituisce una reale restrizione.**

Tipi ammessi da Automation

Pascal type	OLE variant type	Description
Smallint	VT_I2	2-byte signed integer
Integer	VT_I4	4-byte signed integer
Single	VT_R4	4-byte real
Double	VT_R8	8-byte real
Currency	VT_CY	currency
TDateTime	VT_DATE	date
WideString	VT_BSTR	binary string
IDispatch	VT_DISPATCH	pointer to IDispatch interface
SCODE	VT_ERROR	Ole Error Code
WordBool	VT_BOOL	True = -1, False = 0
OleVariant	VT_VARIANT	Ole Variant
IUnknown	VT_UNKNOWN	pointer to IUnknown interface
Byte	VT_UI1	1 byte unsigned integer

- **Ole Variant è un tipo generico, definito solo a runtime. Parametri di questo tipo possono contenere array degli altri tipi**
- **Assegnando per esempio a un OLE Variant il tipo “array di byte” possiamo avere uno stream in cui far passare strutture complesse**