Componenti ActiveX

Enrico Lodolo
e.lodolo@bo.nettuno.it

Componenti e produzione industriale

- **La produzione del software è ancora un processo di tipo sostanzialmente artigianale**
- **La tecnologia dei componenti ha come obiettivo la definizione di una metodologia industriale per la produzione del software**
- **Il modello a cui ci si ispira è l'industria dell'hardware: i computer vengono costruiti con una struttura modulare che utilizza, a vari livelli, componenti standard.**
- **Possiamo distinguere sostanzialmente 5 livelli:**

	Hardware	**Software**
1.	Gate	Espressioni, variabili, condizioni
2.	Block	Funzioni
3.	Chip	Componenti di interfaccia
4.	Card	Business components
5.	Rack	Processi, Applicazioni

- **I primi due livelli sono implementati dai linguaggi di programmazione**
- **Nel 3° e il 4° livello troviamo quelli che vengono comunemente indicati come componenti software**

Modello PEM

- **Un componente è un "circuito integrato" software che comunica con l'esterno attraverso una serie di "piedini"**
- **Un' applicazione in grado di incorporare componenti viene definita *container* ed è l'equivalente software di una scheda elettronica**
- **Abbiamo tre tipi di "piedini": proprietà, metodi, eventi**
  - **Proprietà** (property): "piedini di stato",pseudo-variabili che consentono di agire in modo protetto sullo stato interno
  - **Metodi**: "piedini di ingresso", comandi che provocano l'esecuzione di azioni
  - **Eventi**: "piedini di uscita", provocano l'esecuzione di metodi nel container (callback) in seguito a qualcosa che si verifica nel componente
- **Il modello basato su questi tre concetti viene comunemente chiamato PEM ed è basato sul conecett di incapsulamento**
- **Le tecniche di riuso basate su incapsulamento vengono definite "black-box"**

Serializzazione e introspezione

- **L'idea di componente è strettamente legata a quella di "application builder" ovvero di un ambiente di sviluppo in grado di utilizzarli in modo efficiente**
- **La caratteristica più importante degli application builder è il cosiddetto "design time" ovvero una situazione in cui è possibile incorporaree componenti ed interagire dinamicamente con essi, in modo da definirne aspetto e comportamento (stato iniziale).**
- **All'avvio dell'applicazione i componenti vengono attivati con lo stato iniziale definito a design time**
- **Per supportare questa modalità di funzionamento abbiamo bisogno di altre due caratteristiche:**
- **Introspezione (o riflessione): capacità di fornire una descrizione delle proprie caratteristiche (proprietà, metodi ed eventi) in modo da consentire l'interfacciamento dinamico e l'"esplorazione"**
- **Serializzazione (o persistenza): capacità di memorizzare il proprio stato in uno stream e di ricaricarlo successivamente. la serializzazione consente la memorizzazione dello stato a design time e il successivo ripristion a run-time**

UI Components e “in-place activation”

- **I componenti non sono necessariamente elementi di interfaccia utente**
- **Esistono componenti “invisibili” che gestiscono problematiche di comunicazione (p. es. seriale o sockets) o che consentono l’interfacciamento con un database**
- **Se invece un componente è visibile si parla di *UI component***
- **Gli *UI components* devono possedere una capacità in più, ovvero quella di rappresentare se stessi in una finestra di tipo “child” che viene incorporata in un form e di comportarsi come “controlli”**
- **Questa capacità viene definita *in-place activation***
- **Devono anche poter ricevere dall’ambiente una serie di informazioni denominate *proprietà di ambiente*: colore dello sfondo, font di default ecc.**

Componenti: modelli disponibili

- **VBX: (Microsoft) legati al Visual Basic, hanno decretato il successo di questa teconologia (solo 16 bit), stanno cadendo in disuso**
- **VCL: legati agli ambienti Borland (Delphi, C++ Builder), (16/32 bit) - object oriented**
- **JavaBeans: componenti di Java, nascono dalla collaborazioone tra Sun e Borland e derivano dall'esperienza VCL - object-oriented**
- **OCX/ActiveX: basati su COM, indipendenti dal linguaggio (16/32 bit) - object-based**

Componenti ActiveX

- I componenti ActiveX sono basati su COM e cosituiscono in qualche modo il “riassunto” delle tecnologie basate su questo modello
- Si avvalgono della tecnologia dei documenti composti (OLE Documents) per realizzare l’in-place activation e la serializzazione
- Utilizzano Automation per implementare proprietà, eventi e metodi (modello PEM) e le capacità di introspezione
- Sono implementati come server in-process (quindi DLL)
- Inizialmente venivano chiamati OLE COntrols, poi OCX e infine ActiveX Components
- Sono piuttosto complessi da realizzare, richiedono infatti l’implementazione di una ventina di interfacce (non tutte obbligatorie)
- Nel passaggio da OCX a ActiveX sono state introdotte alcune nuove interfacce legate all’utilizzo all’interno dei browser e quindi in ambito Internet/Intranet (security, URL, comunicazione su socket ecc.)
- In pratica non vengono mai realizzati manualmente ma attraverso strumenti messi a disposizione dagli ambienti di sviluppo (Wizards o strumenti che convertono un form in un componente)

In-place activation

- **I componenti ActiveX sono in pratica server OLE in miniatura**
- **Implementano quasi tutte le interfacce OLE Documents:**
  - IOLEObject
  - IOLECache
  - IOLECache2
  - IOLEInPlaceActiveObject
  - IDataObject (per Drag and Drop)
- **A differenza dei normali server OLE (p.es. Excel dentro Word) che funzionano in modalità outside-in gli ActiveX funzionano in modalità inside-out**
- **Outside-in: a un oggetto Excel incorporato in un documento Word è normalmente inattivo 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**
- **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. Non esiste un'immagine statica**

Serializzazione

- **Anche il meccanismo di serializzazione deriva da OLE ed è basato sulla tecnologia “structured storage”**
- **Si utilizza lo stesso meccanismo che permette il salvataggio dell’oggetto Excel nel documento Word**
- **Il componente implementa l’interfaccia IPersistStorage:**

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

- **Il container chiama i metodi Save e Load passando un’interfaccia IStorage. Il componente salva o carica il proprio stato in questo storage**

Modello PEM e introspezione

- **Automation fornisce la tecnologia per implementare sia i "piedini" del nostro circuito integrato software sia l'introspezione**
- **L'introspezione è realizzata dalle *type libraries.* Come abbiamo visto queste forniscono un cpmpleto meccanismo di esplorazione delle capacità di un oggetto COM**
- **Le proprietà e i metodi vengono imlementate sotto forma di un'interfaccia IDispatch esposta dal componente**
- **In pratica le proprietà sono coppie di metodi (accessors) che realizzano la lettura (get) e la scrittura (set) della proprietà**
- **L'implementazione di una dual interface per tale scopo è fortemente consigliata ma non obbligatoria**
- **Il container mette invece a disposizione una interfaccia Idispatch per le variabili di ambiente: in tal modo il componente può accedere a tali informazioni**

Eventi

- **Anche l'implementazione degli eventi si basa su Automation ma il meccanismo è un po' più complesso**
- **Il componente definisce nella sua type library l'elenco degli eventi che è in grado di innescare sotto forma di *dispinterface* ma non implementa questa interfaccia**
- **Il container legge dalla type library la definizione della *dispinterface* e provvede a fornire l'implementazione**
- **A runtime il componente richiede questa interfaccia e la utilizza ogniqualvolta si renda necessarion innescare un evento**
- **Ne gergo COM 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)**
- **In realtà il container non comunica direttamente con il componente ma attraverso oggetti intermedi definiti "connection points" e deve predisporre un altro oggetto intermedio detto "client site" per ogni controllo incorporato**

COM+

- **Annunciato recentemente (settembre 97)**
- **Disponibile verso la fine del 1998**
- **Caratteristiche:**
  - Semplificazione del modello: il supporto run-time viene arricchito in modo da realizzare tutte quelle funzionalità che adesso devono essere implementate dai framework ad oggetti degli ambienti di sviluppo
  - Il meccanismo delle class factories viene implementato dal supporto runtime e sul lato client si vedono dei normali costruttori
  - Marshaling automatico: metadati, superamento del dualismo vtable-dispinterface e aumento dei tipi di dati utilizzabili
  - Ereditarietà (!): interfaccia/implementazione per in-process servers, solo interfaccia per gli altri
  - Interoperabilità con COM attuale

COM+ - Semplificazione

- **Ci sarà una grossa semplificazione per chi realizza i componenti:**
- **Il grassetto evidenzia le parti realizzate automaticamente dal runtime:**

COM	**COM+**
Class Factory	**ClassFactory**
DLLRegister	**DLLRegister**
Reference Counting	**Reference Counting**
QueryInterface	**QueryInterface**
IDispatch	**IDispatch**
Connection Points	**Connection Points**
TypeInfo	Metadata
Methods	Methods