Oggetti o Componenti?
Esperienze nel Settore Bancario

Carlo Pescio
Eptacom Consulting
pescio@acm.org
http://www.programmers.net/artic/Pescio

Oggetti o Componenti?

Brevi osservazioni su alcuni progetti

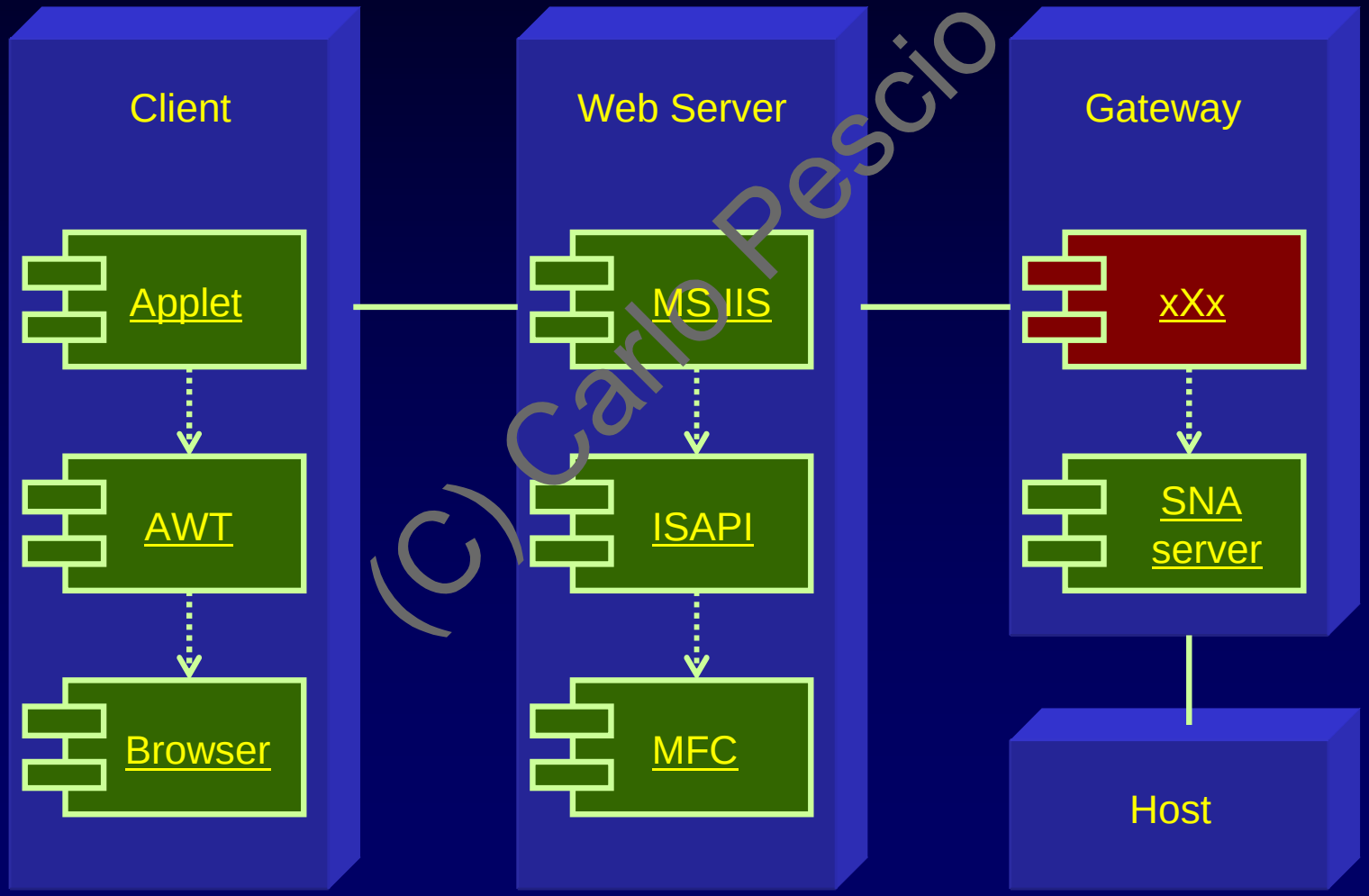
Per mantenere il focus, ho selezionato una famiglia di prodotti (semplici) di remote banking evolutasi negli ultimi quattro anni. Ero coinvolto principalmente in design review e refactoring, mentre seguivo altri progetti per la stessa azienda. Questo spero favorisca un giudizio “neutrale”...

Ogni progetto e’ terminato con successo per qualita’ del prodotto e soddisfazione dei clienti. Non tutti sono terminati nei tempi desiderati (= minori di quelli comunque previsti :-).
Ho riassunto le “lezioni apprese”, scegliendole tra quelle corrispondenti a quanto riscontrato anche in progetti di scala maggiore.

Il risultato potrebbe sembrare un po’ iconoclasta e/o conservatore. Semplicemente, e’ basato su esperienze reali e non sulle infinite promesse...

Remote Banking [V1]

Macro-Architettura



Remote Banking [V1]

Considerazioni sull'assemblaggio

Come spesso accade lavorando con macro-componenti e framework orizzontali ad oggetti, ottenere una versione quasi-funzionante ha richiesto uno sforzo molto contenuto.

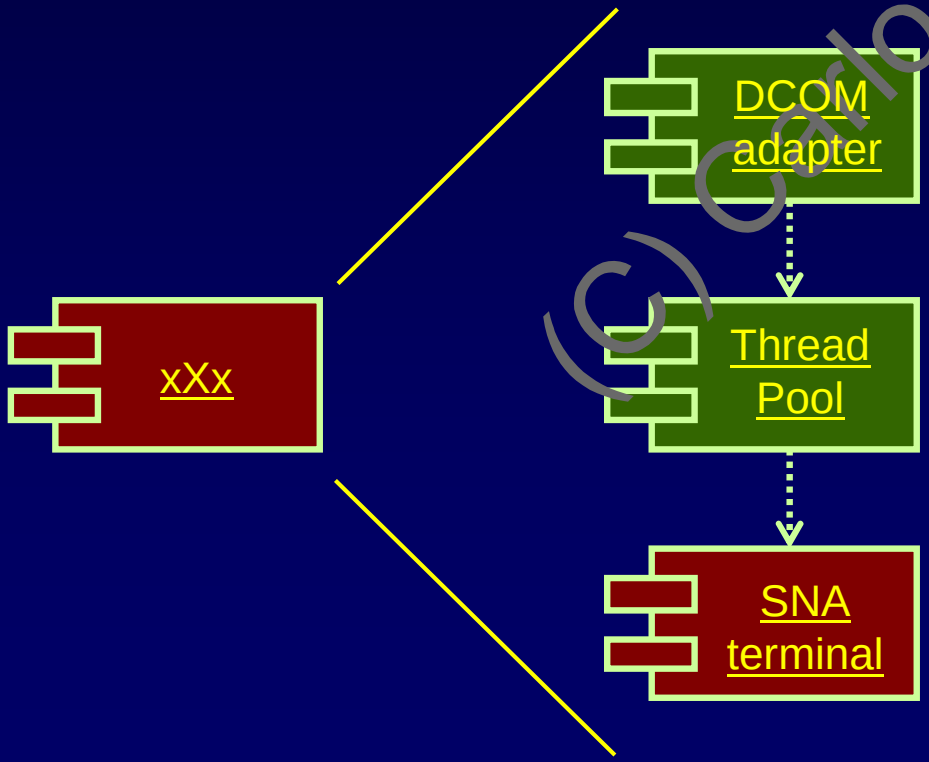
Come altrettanto spesso avviene, raggiungere un livello qualitativo adeguato (soprattutto sul profilo della robustezza) ha richiesto uno sforzo proporzionalmente molto (troppo?) grande.

Vedremo di seguito alcune considerazioni e riflessioni sui problemi e sui risultati ottenuti, prendendo singolarmente in esame i vari macro-blocchi.

Remote Banking [V1]

Gateway: Componente xXx

Responsabile di comunicazione con Host, tramite SNA. Interfaccia piuttosto semplice (logon/exec asincrona/logoff). Gestisce thread pooling interno. Standard DCOM. Acquistato da un [grosso] fornitore italiano, completo di sorgente C++.



Remote Banking [V1]

Componente xXx:

Purtroppo si rivela inaffidabile sotto carico. Il supporto del fornitore e' praticamente nullo: viene accusato il sistema operativo (service pack), poi lo SNA server (altri service pack), poi il middleware (DCOM, RPC). Suggeriscono di inserire dei ritardi per evitare il problema.

Conseguenza:

Si spendono circa due mesi/uomo dentro il sorgente di xXx per identificare e riparare i numerosi difetti (ereditati da SNA terminal). Questo comporta peraltro il coinvolgimento di persone inizialmente estranee al progetto: sia per l'identificazione (competenze diversificate) che per il fix (low-level). Il codice di SNA terminal e' peraltro di pessima qualita' (es. funzione principale di oltre ottomila righe).

Remote Banking [V1]

Considerazioni

Nessuno ha ipotizzato la ricerca di un componente alternativo. Questo sembra essere un pattern ricorrente: avendo una soluzione pseudo-funzionante si investe nella riparazione e non in sostituzioni/re-design.

Fortuna o sfortuna? Il componente era realizzato in C++, linguaggio su cui l'azienda aveva discreta competenza. Questo ha incentivato la ricerca e l'eliminazione dei bug, distogliendo pero' da possibili alternative. Non posso dire cosa avrebbe portato al miglior risultato in minor tempo...

SNA terminal: questo e' un classico caso in cui sarebbe stato meglio (per il produttore ed i suoi clienti) non riusare un componente!

Remote Banking [V1]

Lezioni

I componenti sono Black Box, ma solo finche' funzionano!
In caso di malfunzionamento, il sorgente e' ancora l'unico modo per identificare le cause del fallimento. Acquistare senza sorgente significa fidarsi ciecamente del fornitore.

Corollario: i componenti possono essere scritti in un linguaggio qualunque, ma solo finche' funzionano!
Se il componente xXx fosse stato scritto in un linguaggio estraneo alla cultura dell'azienda, si sarebbe comunque innescato un meccanismo rischioso (ricerca di alternative e/o di personale adeguato per il fixing).

Remote Banking [V1]

Lezioni

Il middleware e' un alleato, ma solo se possiamo fidarci!
Il fornitore di xXx poneva dubbi sul funzionamento del middleware (il DCOM all'epoca era "nuovo", lo SNA server esisteva in diverse versioni con vari service pack), sul sistema operativo, ecc.

Corollario: valutare a fondo i rischi/benefici di uno standard.
In questo caso, se xXx fosse stato un semplice eseguibile con cui comunicare tramite pipe, o una semplice DLL, avremmo perlomeno potuto escludere sin dall'inizio alcune possibili cause di fallimento.
In questo progetto i vantaggi di avere xXx come componente D[COM] erano praticamente nulli!

Remote Banking [V1]

Web Server: ISAPI Extension

La extension DLL e' stata realizzata in C++, usando un framework ad oggetti white-box (MFC).
In genere, un framework white-box richiede al programmatore una certa conoscenza del modello di funzionamento interno.

In questo caso specifico, cio' non ha causato problemi.
Le classi utili per una ISAPI extension sono solo debolmente accoppiate al resto di MFC. Il programmatore lavorava di fatto con un mini-framework specializzato per il suo problema, piu' qualche classe di infrastruttura.

Il risultato e' stato rapido da assemblare e non abbiamo avuto problemi di alcun tipo. Il C++ ha consentito un ottimo livello di gestione delle eccezioni run-time (high reliability).

Remote Banking [V1]

Lezioni

White-box non e' necessariamente "cattivo"

Il reale problema di molti framework white-box e' la pretesa di coprire un dominio troppo vasto, costringendo gli sviluppatori ad un forte investimento nell'apprendimento.

Piccoli framework white-box (domain-specific, problem-specific) possono invece essere adottati con successo in tempi brevi.

L'ereditarieta' di implementazione non e' "cattiva"

Estendere una classe parzialmente implementata tramite ereditarieta' di implementazione impone dei vincoli sul linguaggio di programmazione da utilizzare. In molti casi concreti questo non e' affatto un problema.

Remote Banking [V1]

Client: brevi considerazioni

Sviluppato con AWT, in epoca pre-Beans e pre-Swing. Lo sviluppatore principale aveva (ed ha poi) investito molto tempo per conoscere i dettagli di AWT. Gran parte di questa conoscenza e' ora obsoleta.

Componenti e mercato: uno degli elementi di interfaccia (la "classica" griglia) e' stato acquistato da terze parti.
La ricerca di un componente adeguato ha richiesto un certo sforzo: di norma, i piu' famosi erano anche i piu' sofisticati, ricchi di funzionalita' a noi inutili (es. spreadsheet) e di conseguenza di dimensioni "grandi" (per una applet...).
Il componente "slim" acquistato (con source) ormai non e' piu' supportato. Il produttore ha solo versioni Beans e/o Swing. Il sistema in esame e' pero' ancora in vita!

Remote Banking [V1]

Lezioni

Un framework ad oggetti (specie WB) e' un investimento
Come ogni forma di investimento, occorre la capacita' di tenersi alla larga da quelli senza sufficiente ritorno a lungo termine, o la volonta' di rischiare in cambio di possibili ritorni a breve termine.
Un framework white-box di dimensioni considerevoli andrebbe adottato solo dopo attente valutazioni sulla sua vita probabile.

Componenti [1] Slim (ovvero: la tecnologia non basta!)
Il mercato si adatta alle richieste. Fino a che la scelta avverra' in base alle feature list, avremo software "fat" e componenti "fat".

Remote Banking [V1]

Lezioni

Da cool a legacy: chi mantiene i componenti?

Nella generale speranza che acquistare i componenti da terze parti risolva i problemi di produttivita', sembra che troppi si dimentichino che molte risorse vengono spese per modificare e manutenere i sistemi costruiti.
A che serve essere veloci a fare qualcosa se ogni volta dobbiamo rifarlo (componenti chiusi e non piu' supportati)?
Quanto e' forte l'impegno del produttore nei confronti di chi costruisce software con i loro componenti?

Sorgenti e fiducia

Un altro problema emergente e' quanto possiamo fidarci dei componenti "black box". Sandboxing e' una opzione, ma non sempre percorribile (firma componente Vs firma package).

Remote Banking [V1]

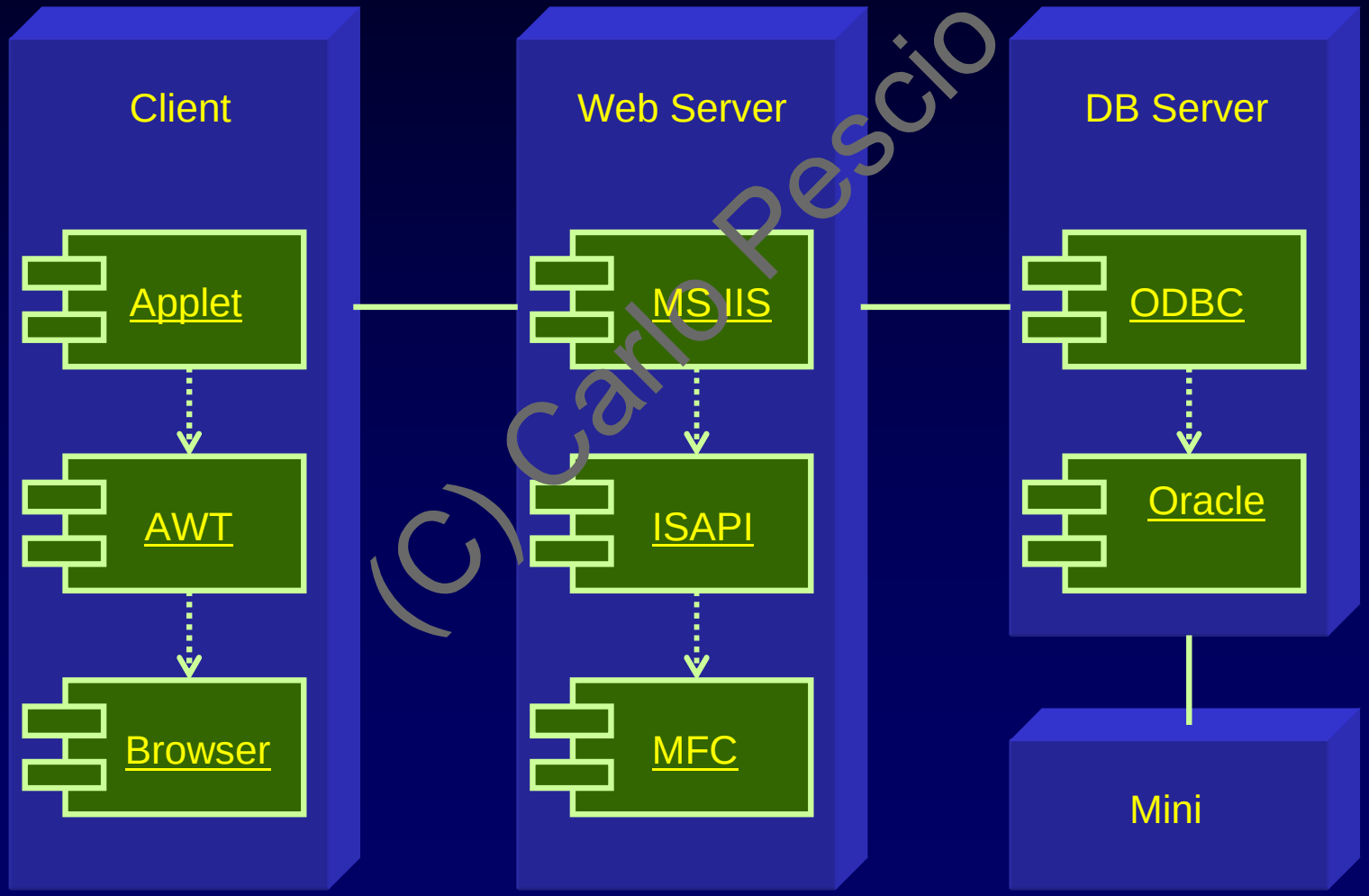
Considerazioni Finali

L’uso di componenti ha portato ad una fase di costruzione molto rapida, ma con forti ricadute negative in termini di tempo speso in acquisizione, debugging e manutenzione dei componenti acquisiti.

Da questo punto di vista gli oggetti hanno dato risultati migliori quando la scelta e’ caduta su piccoli framework (o piccole parti di framework debolmente accoppiate al resto) peraltro mantenuti in vita a lungo dai produttori.
La scelta di framework di ampie dimensioni e’ molto critica, soprattutto se il linguaggio e la tecnologia sottostanti sono in pieno fermento e quindi fortemente instabili (vedi continua crescita delle librerie Java).

Remote Banking [V2]

Macro-Architettura



Remote Banking [V2]

Brevi note descrittive

Pensato per piccoli istituti/filiali che volevano offrire un servizio simile al precedente. In pratica, sostituisce il Gateway + Host con un DB Server, cui accede concorrentemente anche un mini per mantenere integrita' tra database su DB Server e quello di istituto/filiale.

Riuso:
La parte client, come ragionevole, e' stata completamente riutilizzata, senza alcuna modifica.
Teoricamente avremmo potuto riusare l'ISAPI del Web Server, semplicemente creando un componente xDBx con la stessa interfaccia di xXx. In pratica, alcune forti differenze (es. sincrono Vs asincrono) hanno inizialmente scoraggiato il responsabile del progetto.

Remote Banking [V2]

DB Server: configuration hell

ODBC e’ (in teoria) un classico esempio riuscito di tecnologia a componenti black-box: (sempre in teoria) possiamo installare un driver ODBC adeguato e utilizzare lo stesso codice con database diversi (es. Oracle o SQL Server) o release diverse del database.

In pratica, trovare un driver ODBC “giusto” per Oracle ha richiesto settimane. Pur re-implementando a mano un livello di thread pooling e sopperendo manualmente ad un problema di thread affinity presente sia in MFC che nei driver, alla fine l’unica procedura “safe” e’ risultata:

- Installare driver fornito da MS per Oracle
- Installare driver fornito da Oracle
- Disinstallare driver fornito da Oracle (!)

Remote Banking [V2]

DB Server: MFC blues

Il layer sovrapposto da MFC ad ODBC e' piuttosto primitivo, cosa indirettamente riconosciuta dall'introduzione di un Wizard nell'ambiente di sviluppo, che genera il codice di classi wrapper relative alle tavole.
Purtroppo MFC si rivela inadatta a creare codice realmente exception-safe per l'accesso al database. Mentre questo puo' essere tollerato su programmi client (si perdono risorse, non si va in crash) e' inaccettabile sul lato server.

Questo ha portato alla riscrittura di un wrapper "leggero" ad oggetti sopra ODBC, con conseguente investimento di tempo. Di nuovo, non e' stato preso in considerazione il semplice rimpiazzo con una ulteriore libreria: in fondo, l'astro nascente di ADO sconsiglia investimenti ulteriori.

Remote Banking [V2]

Lezioni

I componenti richiedono un sistema di installazione serio!

Proprio per l'impossibilita' di capire in modo adeguato le cause dei problemi (in assenza del sorgente), il rischio e' di abbandonare il purgatorio del fatware per l'inferno delle configurazioni. In questo senso, il passaggio a soluzioni standard come [D]COM o CORBA e' una evoluzione interessante, almeno sulla carta (email da Microsoft mentre scrivo queste note: ADO 2.0 ha problemi con IIS 3.0 :-).

Sorgente come risolutore di problemi

Troppo spesso si pensa che avere il sorgente significhi avere un problema. Di fatto, il sorgente puo' semplicemente aiutare a capire dove sta il problema (es. MFC ed eccezioni), ed a scegliere una strada alternativa.

Remote Banking [V2]

Client side

Il programmatore responsabile del lato client ha richiesto il tempo per “rifarlo” con i Java Beans. Su richiesta dei benefici auspicati, l’unico rilevante per il business era la possibilita’ per i clienti di customizzare l’interfaccia in modo semplice. Viste le richieste reali, si e’ concluso che per gran parte di questo bastava qualche parametro in piu’ nella configurazione.

Lezione

Non sempre dobbiamo percorrere l’ultima onda della tecnologia per ottenere benefici concreti. Un focus sul cliente e sulle reali esigenze puo’ far valutare meglio la necessita’ di investire in soluzioni “di avanguardia”.

Remote Banking [V2]

Evoluzione

Con l'arrivo delle ASP e' sembrato utile poter [opzionalmente] rimpiazzare le ISAPI con pagine attive custom. Cio' e' semplice nella prima configurazione (DCOM-based) ma non in questa (ODBC e' chiamato dalla ISAPI).

A questo punto si ripropone l'idea di standardizzare le interfacce verso Host e verso DB, con il requisito ulteriore di una semplicita' di chiamata da ASP, che spinge verso una soluzione sincrona con timeout.

La revisione ha un impatto su gran parte del sistema, escluso il client: le due ISAPI vengono ridotte ad una, si aggiunge un componente DCOM verso ODBC, si modifica interfaccia e quindi implementazione di xXx.

Tuttavia il tempo richiesto per la realizzazione e' breve, dopo una progettazione accurata delle nuove interfacce.

Remote Banking [V2]

Lezioni

Come gli oggetti, anche i componenti non sono una scusa per non progettare. Il design delle interfacce e' basilare. Ma che dire delle interfacce che "compriamo"?
Comprare un componente significa comprare anche la sua interfaccia. Per quanto si possa intervenire con adapter esterni, non sempre si puo' sperare di ottenere un buon risultato globale semplicemente integrando soluzioni "buone" in un'ottica parziale.
Progettare un buon programma significa spesso progettare una famiglia di programmi [Parnas]. Questo e' abbastanza comune nel paradigma ad oggetti (framework), ma di solito richiede una progettazione accurata delle interfacce. Come questo si sposi con l'idea di acquistare i componenti come COTS e' un grosso interrogativo.

Remote Banking [V2]

Lezioni

Nei componenti black-box, il black ha un suo fascino :-)
Durante la ristrutturazione, un programmatore ha unificato le classi di log (la prima proprietaria di xXx, la seconda parte dell'ISAPI di questa versione, che era stata riusata da altri progetti) e ha trasformato il risultato in componente DCOM.
A questo punto, quasi tutti si sono messi a usare quel componente nei vari progetti. Cio' e' relativamente strano perche' il riuso inter-progetto era piuttosto basso in azienda,
ed i vari team avevano gia' delle loro funzioni/classi di log.
La stessa esperienza si e' ripetuta in altre aziende: una volta che una funzionalita' si congela in un componente binario, questa viene percepita come piu' stabile e riusata piu' volentieri. Gli effetti psicologici sono talvolta piu' importanti di quelli tecnologici nella cultura del riuso...

Remote Banking [V2]

Considerazioni Finali

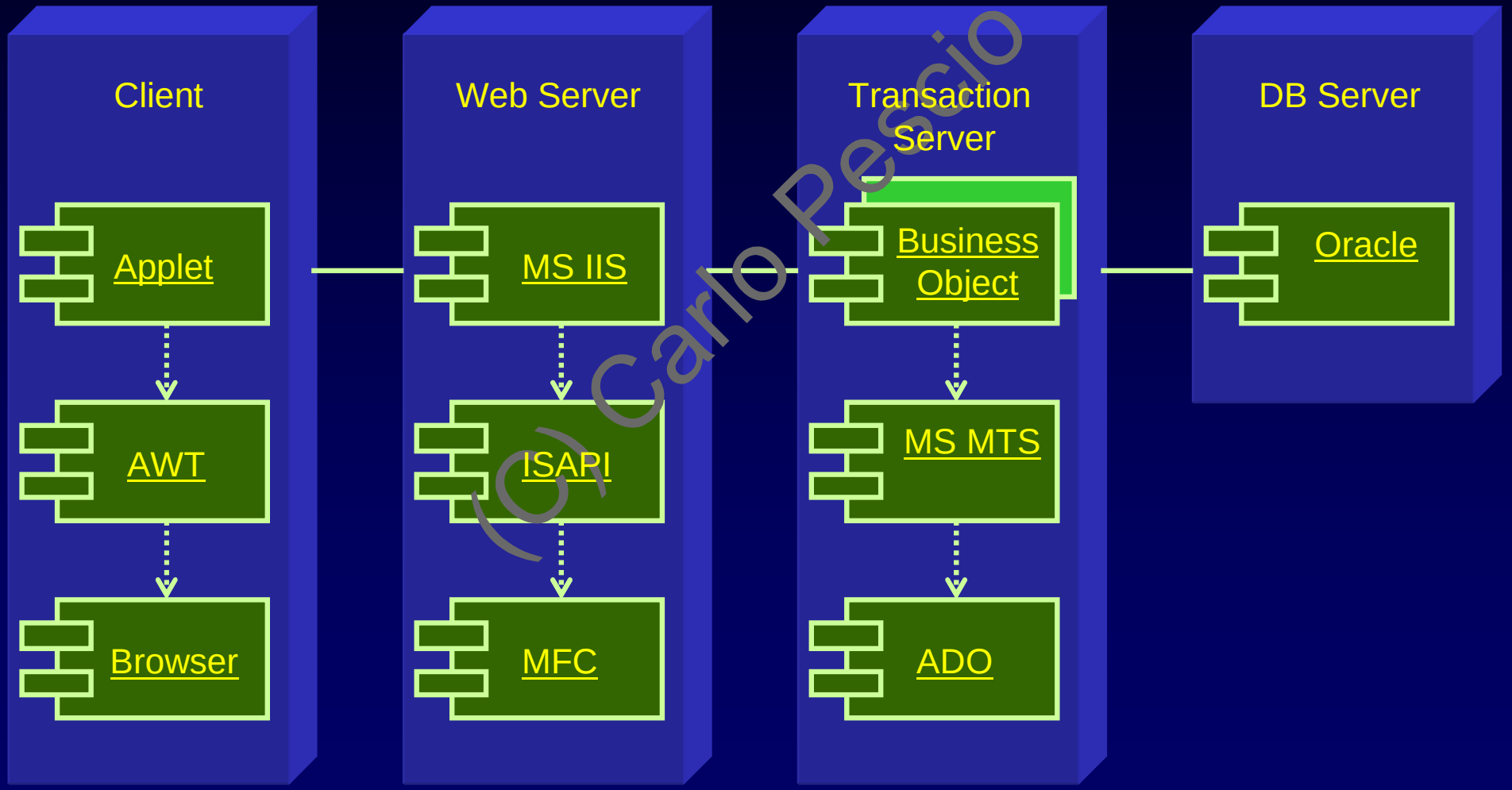
Nuovamente, la relativa rapidita' di assemblaggio della soluzione iniziale si e' rivelata controproducente a medio termine: il componente e' stato integrato as-is, senza fermarsi a ripensarne l'interfaccia in un'ottica di futura estendibilita'. Cultura di integrazione [1] cultura del design.

Del resto, quando si e' rivelata necessaria, la modifica all'interfaccia ha richiesto un intervento interno al componente. Viceversa, la parte ISAPI era basata su classi di granularita' inferiore che hanno assorbito meglio l'impatto della modifica.

Progettare un framework (o famiglia di applicazioni) con componenti plug-in richiede un alto livello di controllo sulle interfacce dei componenti, in contrasto con l'idea di COTS.

Remote Banking [V3]

Macro-Architettura



Remote Banking [V3]

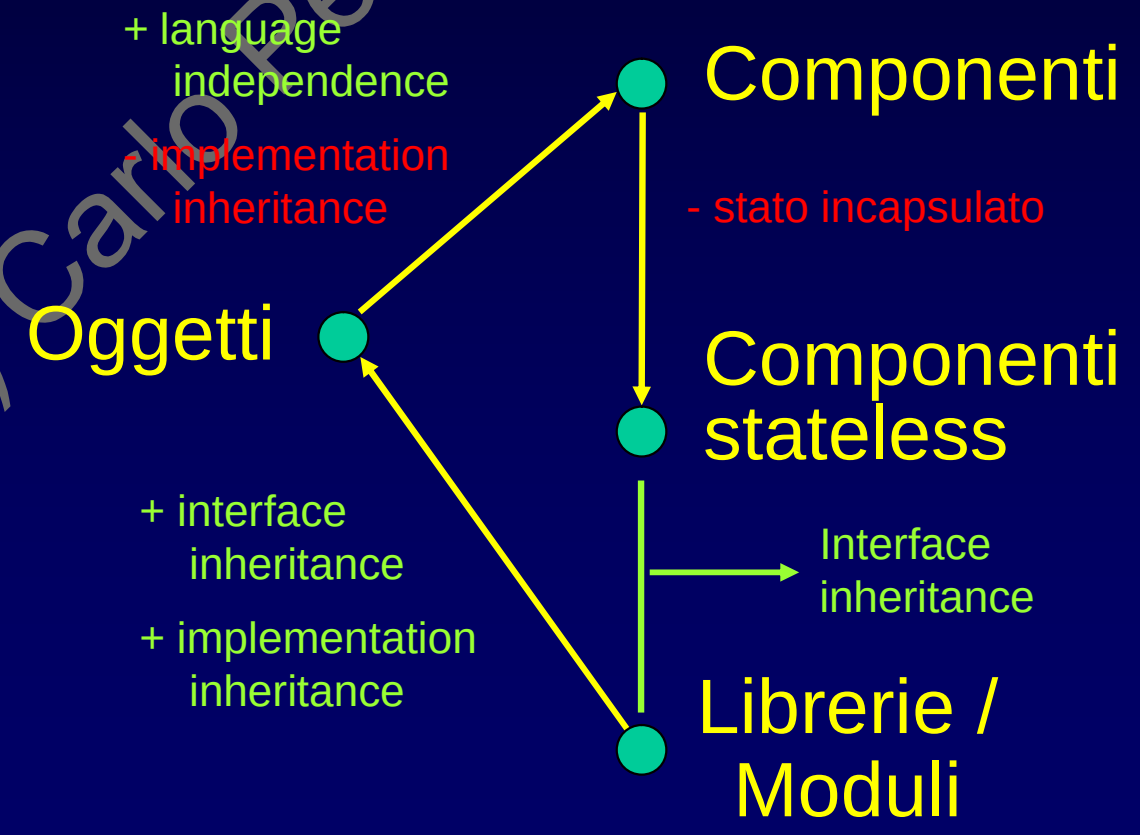
Brevi note descrittive

Introdotto come “versione scalabile” (in teoria) del sistema precedente. Inoltre doveva eliminare la gestione manuale del thread pooling, lasciandola ad MTS.
La trasformazione e’ risultata relativamente semplice: alcune classi preesistenti sono diventate dei componenti [stateless, vedi oltre]. Lo strato ad oggetti da noi interposto verso ODBC in precedenza si e’ adattato con facilita’ ad una interfaccia verso ADO. In tempi successivi, ha consentito di saltare il livello ADO usando direttamente OLEdb, senza modifiche ai business object. ADO/OLEdb si rivela un po’ immaturo, ad esempio i parametri di apertura dei recordset devono essere variati se cambia il DB server. Le prestazioni non sono esaltanti se confrontate con la soluzione precedente (50%).

Remote Banking [V3]

Componenti “stateless”

MTS utilizza un modello di “componenti” privi di stato, per ragioni interne di pooling e di distribuzione del carico.
In effetti, un componente [D]COM privo di stato interno si riduce ad una semplice famiglia di funzioni, strutturata in una interfaccia, senza incapsulazione.



Remote Banking [V3]

Conseguenze

I Business Object devono in qualche modo “estrernalizzare” il loro stato (che avevano nella versione “ad oggetti”).
La prima soluzione adottata dai programmatori finiva per esporre nell’interfaccia molti dettagli implementativi, rendendo il business object stesso difficile da rimpiazzare.
Modificando il design con una soluzione “a Memento” abbiamo ottenuto un buon risultato complessivo.

L’efficienza e la scalabilita’ sono risultate inferiori ad una soluzione custom come la precendente con un bilanciamento mirato. Tuttavia, lo sviluppo e’ decisamente piu’ semplice e rapido, non richiede conoscenze low-level, e presso i clienti ha uno “status” migliore.

Remote Banking [V3]

Lezioni

Componenti stateless » librerie!

Ogni cambio di modello richiede nuovi idiomi e pattern. Il salto complessivo da oggetti a componenti stateless e' notevole, e senza cautele si rischia di perdere i vantaggi sinora acquisiti (es. resilienza da incapsulazione!).
NB: nuovamente, solo [ri]progettazione accurata dell'interfaccia ha risolto i problemi.

Mini-framework ben progettati sono sempre vincenti

Il wrapper leggero ad oggetti su ODBC, di fatto un mini-framework black-box, ha ben sopportato il passaggio ad ADO dello strato di implementazione.

Remote Banking [V3]

Lezioni

Black box = programmazione esplorativa?

Di nuovo, componenti totalmente black-box (OLEdb) hanno richiesto un tuning dei parametri sostanzialmente del tipo “trial and error”. Si dice che con gli oggetti si corre spesso il rischio di dover accedere ai sorgenti. Con i componenti si corre spesso il rischio di non poter accedere ai sorgenti!

I rimedi sono sempre i soliti: specifiche migliori, documentazione migliore. Ma il rischio e’ che nessuna delle due migliorera’ sensibilmente nel medio termine.

Considerazione: quanto sviluppo RAD-incrementale e’ in realta’ un trial-and-error “nobilitato”?

Semplicita’ e’ spesso l’antitesi di efficienza

Risultato complessivo con MTS “inferiore” a sistema custom.

Oggetti o Componenti?

Al di la' dei semplici esempi che abbiamo visto molto brevemente, la mia opinione e' che al momento il passaggio dagli oggetti ai componenti ha senso solo a macro-livello, e solo se siamo in grado di progettare accuratamente le loro interfacce.

Allo stato attuale, i componenti non risolvono in modo completo il problema dell'accesso ai sorgenti, anzi, in molte situazioni lo rendono ancora piu' evidente.
Un passo in avanti sarebbe la definizione di un contratto comportamentale per ogni componente. Ma questo non basta!
Cosi' come nell'hardware ogni IC complesso ha una serie di pin "di test", che in pratica offrono delle viste sugli internals, probabilmente un componente complesso dovra' consentire qualche forma di visibilita' interna a chi lo usa, per identificare eventuali problemi in fase di integrazione.

Oggetti o Componenti?

Gli oggetti sembrano tutto sommato una tecnologia piu' matura, dove benefici, rischi e rimedi sono meglio compresi e privi di illusioni del tipo "componente = bug free".
Oggetti riuniti in piccoli framework white-box possono essere usati con successo anche per lo sviluppo di un singolo componente, con buoni risultati di riuso ed estendibilita'. Riuniti da framework (principalmente) black-box, possono essere usati con successo come base architetturale di un progetto, ovunque non sia necessario un supporto cross-language. Bassa granularita' non e' necessariamente un male (testing, debugging, rimpiazzo)!

Infine, i componenti sembrano avere effetti extra-tecnici interessanti (stimolo al riuso) ma la la prospettiva di manutenzione di macro-componenti (anziche' micro-oggetti) acquisiti da terze parti rimane un po' preoccupante.

? ? ? ?

Opinioni, domande, ripensamenti, giudizi...

Carlo Pescio

pescio@acm.org

http://www.programmers.net/artic/Pescio