



F.O.A.M.

Free Object Access Method

NET Services

Premessa: che cosa è il Middleware?

Gli ambienti di calcolo tradizionali sono di natura eterogenea, essendo formati da un mix vario di *hardware*, sistemi operativi, tecnologie di rete e linguaggi di programmazione.

I motivi di questa situazione sono solitamente storici: qualsiasi cosa che è stata pensata per essere più importante o "migliore" in un dato periodo, è finita per essere aggiunta all'ambiente di elaborazione. Ad esempio, alcuni anni fa Java ancora non esisteva, mentre oggi difficilmente qualcuno immagina un ambiente distribuito senza che almeno alcuni dei codici siano scritti in Java.

Prevedibilmente, gli ambienti eterogenei saranno con noi per il prossimo futuro e le società sono da sempre forzate ad abbracciare le *hot technologies*, come Java, HTTP, XML o ATM.

La necessità di integrare le applicazioni nasce dal conflitto insito negli ambienti eterogenei. Varie parti dell'ambiente devono essere in grado di comunicare liberamente fra loro, nonostante che le varie tecnologie non sono, di solito, progettate per essere integrate. L'esplosione recente di Java, di *e-commerce* e del Web ha reso abbondantemente chiaro che oggi è molto difficile essere competitivi in questo settore a meno di utilizzare componenti software integrati a tutti i livelli: dal *browser* Web dei clienti, fino ai sistemi di controllo *back-end*.

Proprio l'integrazione di applicazioni in un ambiente eterogeneo è estremamente difficile. La larga varietà di linguaggi di programmazione, tecnologie di rete, sistemi operativi e altro, pone formidabili problemi tecnici, mentre la complessità e il costo di soluzioni di integrazione personalizzate non sono commercialmente plausibili, se non per le società più grandi.

Il *middleware* focalizza la sfida di integrazione fornendo un comune substrato di comunicazione non fine a se stesso, ma integrato in modo coerente. In effetti, il *middleware* agisce come "colla di integrazione" e consente agli sviluppatori di concentrarsi sulla logica dell'applicazione piuttosto che costruire l'infrastruttura di comunicazioni.

Il protocollo: FOAM.

FOAM è un protocollo, quindi è semplicemente un pezzo di carta su cui sono riportate delle regole. Nel caso specifico, FOAM (Free Object Access Method), è un protocollo che descrive come trasferire informazione tra diversi applicativi ed invocare procedure o metodi remoti. FOAM trae ispirazione da CORBA e SOAP, ma semplifica notevolmente il compito dello sviluppatore, infatti si basa su protocolli standard per il trasporto dell'informazione e su un qualsiasi formato testuale per la codifica della stessa. Conviene ricordare che anche XML (su cui è basato SOAP) è un formato testuale, quindi FOAM, all'occorrenza, supporta anche XML, lasciando libero il formato. FOAM di base prevede un meccanismo completamente asincrono. Lo sviluppatore è libero di inserire elementi di sincronizzazione tra il metodo chiamante e quello chiamato. Grazie a FOAM ed alla particolare codifica adottata, è possibile l'invocazione multipla di più metodi remoti, dislocati su macchine diverse. FOAM supporta, a livello di protocollo, caratteristiche di **multicanalità**, *fault tolerance*¹ e *load balancing*², implementate in un modo del tutto nuovo: tradizionalmente più server collaborano nell'erogazione dei servizi. FOAM segue una nuova filosofia, nella quale i server competono su una risorsa virtuale che è l'erogazione del servizio. Tale approccio (in linea con la filosofia completamente asincrona propria del protocollo) consente:

- un notevole abbattimento del traffico di rete generato dai partner;
- **il funzionamento dei client anche quando l'Application Server risulta spento** (ovviamente in questo caso ai client non verranno erogati i servizi richiesti finché non "risale" almeno un Application Server);
- **l'accesso alla rete Internet anche in presenza di un firewall.**

L'idea alla base di FOAM è che la comunicazione tra le parti coinvolte nel processo debba avvenire in modo asincrono, infatti uno scambio sincrono porta con sé una serie di problemi:

- i sistemi coinvolti devono essere disponibili e funzionanti nel dato istante di tempo in cui lo scambio avviene;
- durante le operazioni, in genere, i sistemi coinvolti hanno un picco di carico;
- se la singola azione di una procedura sincrona fallisce, l'intera procedura fallisce (anche se è sempre possibile prendere accorgimenti affinché ciò non accada);
- la velocità complessiva del sistema è pari alla velocità della periferica più lenta (perché le periferiche veloci devono attendere la risposta di quelle più lente).

Invece l'integrazione fra sistemi quanto più è mission critical, tanto più dovrebbe essere effettuata in modo asincrono. Vale a dire che i messaggi che gli attori si scambiano devono essere depositati in un gestore di code e in questo modo definitivamente disaccoppiati dai sistemi che tali messaggi utilizzano e/o creano.

I messaggi costituiscono l'elemento atomico dello scambio, sono il supporto attraverso il quale avviene la transazione; contengono:

- dati;
- comandi per l'esecuzione di metodi remoti;
- oggetti binari (ad esempio gli Adapter).

¹ si dicono Fault-Tolerant o tolleranti ai guasti, i sistemi capaci di non subire fallimenti anche in presenza di guasti hw o sw;

² per load balancing si intende la capacità di un sistema distribuito di bilanciare il carico elaborativo tra i vari agenti computazionali, al fine di migliorare l'efficienza complessiva del sistema.

I messaggi (propriamente detti DOCUMENTI FOAM) devono avere una serie di proprietà (non necessariamente tutte insieme):

- Lifetime: è l'intervallo di tempo in cui il documento ha significato (è potente). Fuori dall'intervallo di tempo, il documento è ignorato;
- Identity: codice univoco che identifica il documento. Grazie a questa proprietà è possibile, ad esempio, salvare i documenti in un database relazionale ed ottenere una serie di caratteristiche interessanti, quali la persistenza degli oggetti, il log sulle transazioni, ecc.;
- Acceptance: un documento è "accettabile" solo dai nodi riceventi che instradano verso il destinatario di quel documento o delle sue copie (basandosi sull'identity). Un esempio: se un documento contiene codice binario (un adapter) per Windows, non dovrà essere accettato da macchine Unix/Linux e viceversa;
- Idempotence: la capacità di un documento di essere ricevuto e trasmesso più di una volta, avendo sui sistemi coinvolti lo stesso effetto che avrebbe se fosse trasmesso una sola volta;
- Receipts: ogni documento raccoglie le ricevute dei sistemi che attraversa. Le ricevute possono essere di tipo "delivery" (quelle raccolte mentre viene instradato verso la sua destinazione) o di tipo "commitment" (certificano che il documento è stato ricevuto dal destinatario e che il suo contenuto è stato capito e applicato).

Sincronizzazione

La sincronizzazione è un attributo sia dei processi, sia della comunicazione.

Si dice che un sistema è sincrono se soddisfa le seguenti proprietà:

1. Esiste un limite superiore al ritardo del messaggio;
2. Il tempo necessario al processo per eseguire uno step è conosciuto a priori.

In un sistema sincrono è possibile misurare il timeout dei messaggi, e questo fornisce un meccanismo per determinare i failure. Un sistema invece è asincrono se non esiste un limite al ritardo del messaggio, oppure al tempo necessario ad un processo per eseguire uno step. Perciò, per dire che un sistema è asincrono non deve essere fatta nessuna assunzione sui tempi.

Il modello asincrono ha ottenuto molto più interesse perché ha una semantica relativamente semplice; **le applicazioni sviluppate su questo modello sono portabili in modo più agevole rispetto ad applicazioni che fanno delle assunzioni sui tempi.**

Ovvero, nella comunicazione asincrona il canale contiene logicamente un certo numero di buffer dello stesso tipo, in cui vengono ordinati i messaggi, successivamente inviati secondo una certa disciplina: tipicamente si tratta di una coda FIFO. L'esecuzione di una send non provoca l'attesa da parte del mittente che il destinatario abbia effettuata la receive corrispondente. Ciò permette al modulo mittente di svincolare, entro certi limiti, il proprio funzionamento da quello del destinatario. In alcuni casi ciò può essere importante per evitare situazioni di stallo e per **aumentare la velocità di elaborazione.**

Nella comunicazione sincrona i canali non contengono logicamente alcun spazio di bufferizzazione. In questo caso i moduli mittente e destinatario stabiliscono una sorta di rendez-vous.

I modelli sincroni ed asincroni sono i due estremi di uno spettro di possibili modelli.

La tecnologia: i NET Services.

Un programma viene realizzato per rispondere ad una determinata esigenza, quindi offre uno o più servizi. Gli utenti di un programma sono gli esseri umani che hanno la necessità di fruire del/dei servizi offerti dal programma.

I NET Services nascono con l'ambizione di dare una risposta alle seguenti domande:

- cosa succede quando il fruitore di un servizio non è un utente, ma un altro programma?
- come è possibile rendere fruibile un servizio indipendentemente dal protocollo, dalla piattaforma, dal Sistema Operativo e dal linguaggio di programmazione utilizzato?

Le caratteristiche principali dei NET Services possono essere così riassunte:

- sono accessibili attraverso un protocollo standard;
- utilizzano un formato testuale (eventualmente XML) per lo scambio dei dati;
- la scelta di tecnologia di network è trasparente sia al provider sia all'utilizzatore del servizio (requestor);
- consentono l'utilizzo di un'unica tecnologia di integrazione dei sistemi sia all'interno sia all'esterno di un firewall;
- supportano l'integrazione dei servizi indipendentemente dalla piattaforma utilizzata dal provider, consentendo il riutilizzo delle preesistenti infrastrutture di back-end;
- supportano la multicanalità: occorre svincolarsi dall'idea che il contesto di networking sia solo ed esclusivamente la Rete. I NET Services sono progettati e realizzati in modo da prevedere più canali di networking (WAP, Internet, SMS, Fax, ecc.).

L'architettura alla base dei NET Services poggia sull'interazione di tre attori principali:

- il *service provider*: il server che fornisce il NET Service;
- il *service registry*: un registro che consente di reperire i descrittori del servizio (file contenente i dettagli dell'interfaccia e dell'implementazione del servizio, tra cui i tipi di dati utilizzati, le operazioni fornite e le informazioni necessarie alla localizzazione del servizio sulla rete);
- il *service requestor*: l'applicazione che richiede il servizio.

E' bene osservare che uno stesso programma può agire contemporaneamente da service provider, service registry e service requestor, relativamente o meno allo stesso servizio.

L'interazione fra i tre attori prevede le seguenti attività:

- **Pubblicazione**: affinché un NET Service sia accessibile, è necessario che sia pubblicato per esso un descrittore che indichi al service requestor come interagire con esso;
- **Ricerca**: attraverso la quale il service requestor recupera un descrittore per il servizio o direttamente o richiedendolo al service registry;
- **Bind**: con l'operazione di Bind, il service requestor invoca o inizia un'interazione con l'implementazione del NET Service ospitato sul server provider.

Uno scenario tipico di un NET Service potrebbe essere il seguente:

1. Un service provider ospita un modulo software accessibile mediante la rete;
2. Il service provider definisce un descrittore del servizio per il NET Service e lo pubblica presso un service registry o direttamente presso un service requestor;
3. Il service requestor usa un'operazione di ricerca per recuperare il descrittore del NET Service ed usa tale descrittore per effettuare la bind al service provider ed interagire con l'implementazione del NET Service.

Sono previste due modalità principali di interazione tra service requestor e NET Services, riferite come: modalità RMI (Remote Method Invocation) e modalità Document-Oriented. La prima coincide con la classica modalità di accesso ai servizi fornita dai tradizionali middleware quali EJB, CORBA, DCOM ecc. Il client richiede uno specifico servizio ed attende il completamento dell'operazione da parte del server. La modalità Document-Oriented, invece, è simile all'interazione che si ottiene mediante i protocolli asincroni. In questo caso quando un client effettua una richiesta al servizio, attiva un flusso di elaborazione, ma non ne attende la conclusione.

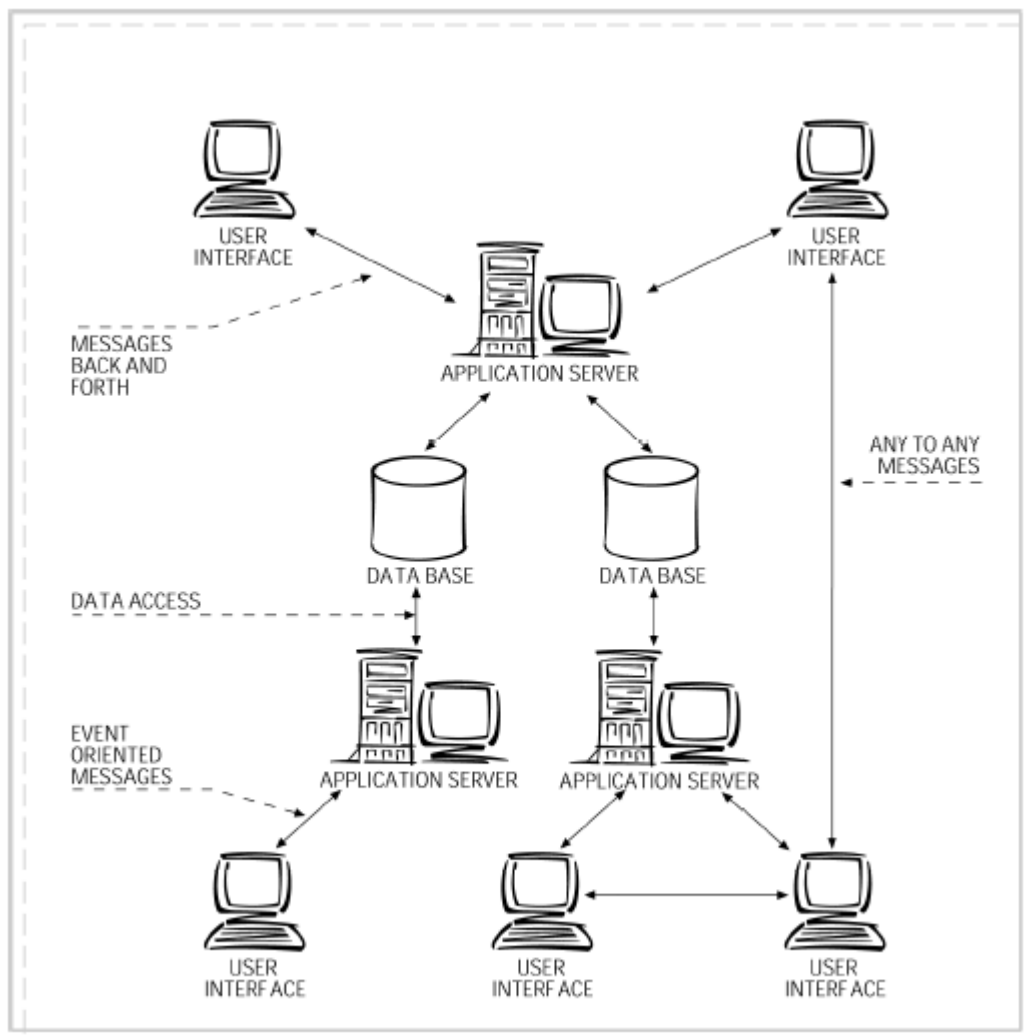


Fig.1: esempio di applicazione a 3 livelli in cui gli Application Server implementano fault tolerance e load balancing.