Università degli Studi di Padova

Processi e concorrenza

SCD

Anno accademico 2006/7
Corso di Sistemi Concorrenti e Distribuiti

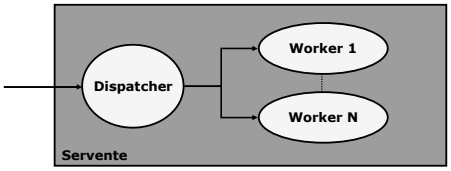
Tullio Vardanega, tullio.vardanega@math.unipd.it

Cliente e servente concorrenti – 1

- **Lato cliente**
  - **La concorrenza interna può mitigare l'effetto del ritardo di rete indotto dalle comunicazioni implicate dall'interazione distribuita tra cliente e servente**
  - **P.es.: a un *Web browser* (lato cliente) conviene eseguire in parallelo**
    - Attivazione della connessione TCP/IP → operazione bloccante
    - Lettura ed elaborazione dei dati in ingresso → può operare in *pipeline*
    - Trasferimento su video → può operare in *pipeline*
  - **Grazie alla concorrenza interna il cliente può anche attuare più sessioni parallele con queste caratteristiche**
    - P.es.: i "*tab*" di Mozilla

Cliente e servente concorrenti – 2

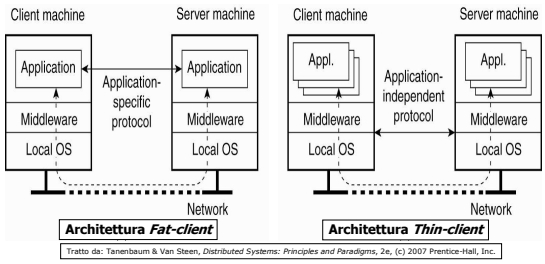
- **Lato servente**
  - **La concorrenza interna offre**
    - Maggiore efficienza prestazionale, ancor più utile e desiderabile che nel cliente
    - Utile modularizzazione (e semplificazione) nella struttura del servente



Problematiche di lato cliente – 1

Trasparenza	Ruolo del MW di lato cliente
Accesso	Fondamentale – *stub* (RPC) o *proxy* (RMI)
Collocazione	Fondamentale
Migrazione / Spostamento	Desiderabile – gestione dinamica delle corrispondenze nome-indirizzo (*naming*)
Replicazione	Utile per nascondere la possibile interazione con più repliche del servente
Concorrenza	Utile (fondamentale dal lato servente)
Malfunzionamento	Desiderabile – p.es. il *caching* del *Web browser*
Persistenza	Non significativa (fondamentale dal lato servente)

Problematiche di lato cliente – 2



Tratto da: Tanenbaum & Van Steen, *Distributed Systems: Principles and Paradigms*, 2e, (c) 2007 Prentice-Hall, Inc.

Problematiche di lato servente – 1

- **Due possibili organizzazioni di servente**
  - **Iterativa → distribuzione verticale**
    - Il servente interpellato soddisfa la richiesta (anche) utilizzando servizi di altri serventi (interni o esterni) e fornisce la risposta corrispondente
    - Nessuna nuova richiesta potrà essere accettata fino al soddisfacimento di quella corrente → per soddisfare più richieste simultanee occorre completa replicazione dell'intero servente
  - **Concorrente → distribuzione orizzontale**
    - Il servente interpellato si occupa solamente di accogliere richieste, demandandone il soddisfacimento a un *thread* o processo distinto
    - Nuove richieste possono essere accolte subito dopo che quella corrente sia stata affidata all'esecutore selezionato
    - Per soddisfare più richieste simultaneamente basta replicare gli esecutori (1 *dispatcher* – N *worker*)

Problematiche di lato servente – 2

- **Localizzazione del servente**
  - **A un servente corrisponde una porta (*end-point*) del nodo sulla quale un processo apposito si pone in ascolto**
  - **Porta preassegnata e nota**
    - Esempio: **IANA** (***Internet Assigned Numbers Authority***) attribuisce porte predefinite a specifici protocolli di livello Applicazioni
    - HTTP:80, FTP:20-1, SMTP:25, … (Vedi http://www.iana.org/assignments/port-numbers)
  - **Porta assegnata dinamicamente**
    - Un *daemon* ascolta su una porta preassegnata le richieste in arrivo per un certo insieme di servizi
    - Le richieste per lo stesso servizio vengono tutte inviate su una porta assegnata dinamicamente della quale viene informato il servente corrispondente
  - **Super-Servente**
    - Quando le richieste di servizio sono sporadiche non conviene mantenere i rispettivi serventi (o *daemon*) inutilmente attivi
    - Un super-servente ascolta **tutte** le porte corrispondenti e per ogni richiesta in arrivo risveglia (o crea dinamicamente) il servente corrispondente
    - P.es. **`inetd`** di UNIX e GNU/Linux

Problematiche di lato servente – 3

- **Interrompibilità del servente**
  - **Modello TCP/IP: la rottura della connessione (p.es. per abbandono del cliente) comporta l'interruzione del servizio**
    - Non immediata, ma garantita, senza confusione con richieste successive
  - **Dati "*out-of-band*": il cliente può richiedere al servente di dare ascolto prioritario ad alcuni dati fuori sequenza ma di maggiore urgenza**
    - Cliente e servente devono intrattenere più di una sottoconnessione logica entro una stessa connessione di servizio
      - Una porta distinta per ogni sottoconnessione
      - Designazione di urgenza nell'intestazione dei pacchetti dati (p.es. TCP)

Problematiche di lato servente – 4

- **Servente senza stato (*stateless*)**
  - **Non tiene conto dello stato del cliente**
  - **Non informa il cliente dei suoi eventuali cambi di stato**
  - **Esempio: lo *Web server* accoglie richieste veicolate da HTTP sulla porta 80, le soddisfa, dimentica il cliente subito dopo, e può cambiare locazione, stato ed esistenza dei propri *file* senza dover informarne alcun cliente**
- **Servente con stato (*stateful*)**
  - **Esempio 1: un *file server* (p.es. NFS) tiene traccia di quali utenti remoti abbiano accesso ai propri *file* locali**
  - **Esempio 2: il *MW* di certi clienti pone *cookie* presso il cliente per fornire al servente informazioni correnti sullo stato del cliente**
    - Validi entro o tra sessioni

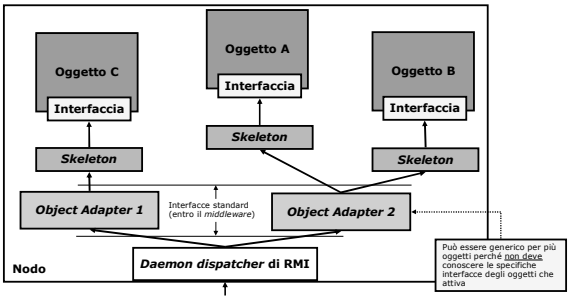
Servente di oggetto – 1

- **Non fornisce alcun servizio di per se ma agisce come tramite di invocazione locale per conto di un cliente remoto**
- **La realizzazione concreta del servente determina se e come l'interfaccia e l'oggetto a lui associati possano essere separati**
- **È desiderabile che il servente di oggetto sia capace di supportare diverse politiche di accesso e di gestione dell'oggetto remoto**

Servente di oggetto – 2

- **Le politiche di attivazione dell'oggetto determinano le modalità con le quali un oggetto remoto può essere invocato**
  - **Per esempio per controllare il caricamento dell'oggetto nello spazio di indirizzamento del servente**
    - Il che equivale all'attivazione dell'oggetto remoto
- **Conviene raggruppare per nodo la gestione di oggetti con la stessa politica di attivazione**
  - **La politica comune viene detta *object adapter* (o *wrapper*)**
    - Oggetti transitori *vs.* oggetti permanenti
    - Invocazione tramite *thread* dedicata (p.es.: *thread pool*)

Servente di oggetto – 3



Migrazione di codice – 1

- **Bilanciamento di carico**
  - **Molto importante nel passato**
  - **Oggi lo è meno vista la potenza di calcolo disponibile**
  - **L'onere di comunicazione è diventato il vero collo di bottiglia**
  - **Migrazione del cliente presso il servente**
    - **Esempio**: un cliente deve effettuare una sequenza complessa di transazioni su base dati ove ciascuna transazione mobilizza grandi volumi di informazione
    - Conviene che la parte coinvolta del cliente operi localmente al servente per limitare il trasporto dati su rete
  - **Migrazione del servente presso il cliente**
    - **Esempio**: un servente che richiede al cliente molti dati da fornire in piccole unità di formato prefissato (p.es: *form*) come prerequisito a una transazione
    - Conviene inviare una parte del servente localmente al cliente per predisporre più velocemente i dati trattati

Migrazione di codice – 2

- **Miglioramento delle prestazioni**
  - **Esempio: parallelizzazione di ricerca su più siti inviando su ciascuno una copia dell'agente di ricerca**
- **Flessibilità di configurazione del sistema**
  - **Fissare staticamente una configurazione è difficile e rischioso quando le condizioni di lavoro non siano note a priori**
    - Ma conviene di più poterla adattare dinamicamente

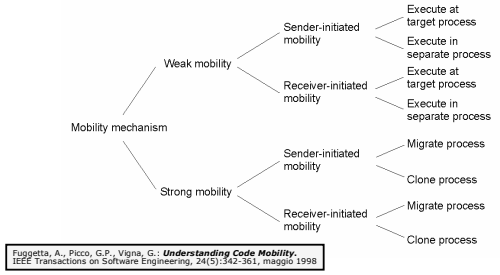
Migrazione di codice – 3

- **Modelli di mobilità (1/2)**
  - **Debole (*weak mobility*)**
    - Solo segmento codice e dati di inizializzazione → il programma migrato riparte sempre dal suo stato iniziale (p.es. Java *applet*)
    - Richiede portabilità del codice, oppure speciale supporto dal compilatore
  - **Forte (*strong mobility*)**
    - Migra anche lo stato di esecuzione → l'esecuzione continua a destinazione
    - Grande complessità realizzativa
  - **Causata dal luogo di residenza iniziale (*sender-initiated*)**
    - Da cliente verso servente → delicato, richiede **autenticazione** del cliente
  - **Causata dal luogo di destinazione (*receiver-initiated*)**
    - Da servente verso cliente (p.es. Java *applet*) → più facile e sicura

Migrazione di codice – 4

- **Modelli di mobilità (2/2)**
  - **Esecuzione nel processo destinatario**
    - **Esempio**: le Java *applet* eseguono nello spazio di indirizzamento del processo *browser* di destinazione (lato cliente)
    - Il processo destinatario deve essere protetto da codice malintenzionato
  - **Esecuzione in processo dedicato**
    - Maggiore protezione da intrusione al costo di maggior onere di comunicazione locale
  - **Clonazione**
    - Simile al modello `fork()` di UNIX, con il processo clonato che eredita codice e stato dal processo clonante

Migrazione di codice – 5



Fuggetta, A., Picco, G.P., Vigna, G.: ***Understanding Code Mobility***. IEEE Transactions on Software Engineering, 24(5):342-361, maggio 1998

Migrazione di codice – 6

- **Modelli di migrazione delle risorse su cui opera il codice mobile (1/2)**
  - **Legame forte (*binding by identifier*)**
    - La risorsa ha identità fisica (p.es. indirizzo IP)
  - **Legame debole (*binding by value*)**
    - La risorsa non ha identità fisica ma solo specifiche caratteristiche attese e dunque può essere copiata a destinazione
      - Esempio: riferimenti a librerie standard in linguaggi come C, C++, Java, Ada
  - **Legame flebile (*binding by type*)**
    - La risorsa deve solo appartenere a un tipo dato fissato che può dunque avere più rappresentazioni
      - Esempio: una stampante PostScript, un dispositivo

Migrazione di codice – 7

- **Modelli di migrazione delle risorse (2/2)**
  - **Risorse mobili (*unattached resources*)**
    - Il legame tra risorsa e nodo è lasco e lo spostamento ha basso costo
      - Esempio: un *file* dati
  - **Risorse legate (*fastened resources*)**
    - Il legame può essere lasco ma lo spostamento è oneroso
      - Esempio: una intera base dati, un intero sito *Web*
  - **Risorse immobili (*fixed resources*)**
    - Il legame è così stretto da non poter essere sciolto
      - Esempio: un dispositivo locale

Migrazione di codice – 8

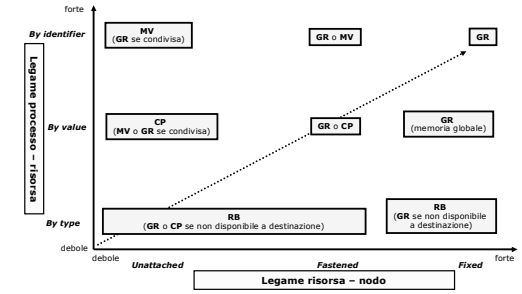
Legame risorsa – nodo (+ debole → + forte)

Legame processo – risorsa (+ forte → + debole)

	Unattached	*Fastened*	*Fixed*
By identifier	**MV** (**GR** se condivisa)	**GR** (o **MV**)	**GR**
By value	**CP** (**MV** o **GR** se condivisa)	**GR** (o **CP**)	**GR** (memoria globale)
By type	**RB** (**GR** o **CP** se non disponibile a destinazione)	**RB** (o **GR** o **CP**)	**RB** (o **GR**)

Cosa fare con le risorse associate a codice che migra
MV: la risorsa può essere spostata
CP: il valore della risorsa può essere copiato a destinazione
GR: la risorsa può essere univocamente riferita da ogni parte del sistema distribuito
RB: la risorsa può essere rappresentata a destinazione da una risorsa analoga disponibile

Migrazione di codice – 9



Agenti *software* – 1

- **Definizione 1**
  - **Unità autonome capaci di eseguire compiti collaborando con altri agenti anche remoti**
- **Definizione 2**
  - **Processo autonomo capace di reagire a cambiamenti nel proprio ambiente e di provocarne, eventualmente in collaborazione con utenti od altri agenti**
- **Caratteristiche salienti**
  - **Capacità di lavoro autonomo e di collaborazione**
    - **Agenti collaborativi**

Agenti *software* – 2

- **Agenti mobili**
  - **Con mobilità forte (più spesso) ma anche debole**
- **Agenti di interfaccia**
  - **Assistono uno o più utenti nell'uso di una o più particolari applicazioni**
    - Esempio: gli *assistant* di molte applicazioni da ufficio
  - **Hanno capacità di apprendimento**
- **Agenti di informazione**
  - **Stazionari: agiscono su informazione in ingresso**
    - Esempio: un filtro di posta elettronica
  - **Mobili: cercano attivamente informazione ove essa risiede**

Agenti *software* – 3

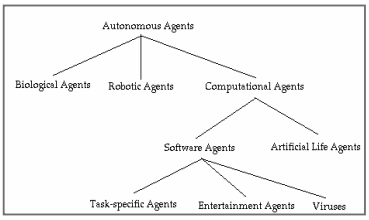
Proprietà	Di tutti?	Capacità implicata
Autonomia	**Si**	**Prendere iniziative proprie**
Reattività	**Si**	**Rispondere prontamente a cambiamenti nell'ambiente**
Proattività	**Si**	**Intraprendere azioni che producano cambiamenti nell'ambiente**
Comunicazione	**Si**	**Scambiare informazioni con utenti e altri agenti**
Continuità	**No**	**Avere un tempo di vita medio-lungo**
Mobilità	**No**	**Poter migrare da un sito all'altro**
Adattività	**No**	**Apprendimento**

Proprietà comportamentali degli agenti *software*

Università degli Studi di Padova

Sistemi distribuiti: processi e concorrenza

Agenti *software* – 4



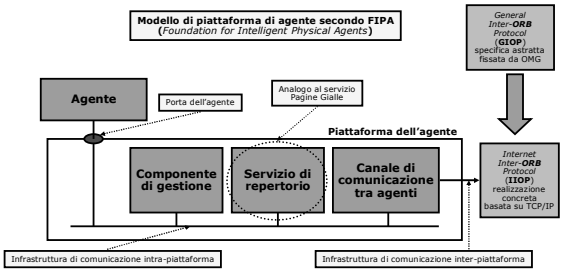
Tratto da: http://www.msci.memphis.edu/~franklin/AgentProg.html

Corso di Laurea Specialistica in Informatica, Università di Padova 25/26

Università degli Studi di Padova

Sistemi distribuiti: processi e concorrenza

Agenti *software* – 5



Corso di Laurea Specialistica in Informatica, Università di Padova 26/26