Un modello di *rendezvous*

SCD

Anno accademico 2007/8
Corso di Sistemi Concorrenti e Distribuiti

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

Modello base – 1

- **Interazione di tipo cliente-servente**
  - **Il processo servente dichiara un certo numero di servizi che esso è disposto a fornire a processi cliente**
  - **La specifica del processo servente dichiara i punti d'accesso (*entry*) corrispondenti a ciascun servizio**
    - Ognuno di essi può prevedere un suo proprio protocollo di scambio parametri
  - **Ciascun processo cliente emette una richiesta d'accesso (*entry call*) nella quale nomina servente e servizio**
  - **Il servente fornisce uno dei servizi richiesti esprimendone esplicitamente la propria accettazione**
  - **La comunicazione tra servente e cliente è sincrona e non richiede necessariamente il passaggio di dati**

Modello base – 2

```
task type Operator is
   entry Query (A_Person    : in Name;
                An_Address  : in Address;
                A_Number    : out Number);
end Operator;
Ann : Operator;
```

Specifica di punto di accesso e protocollo

```
task type User;
task body User is
   My_Number : Number;
begin
   …
   Ann.Query(
      "…", "…",
      My_Number);
   …
end User;
```

```
task body Operator is
begin
   …
   loop
      accept Query(A_Person    : in Name;
                   An_Address  : in Address;
                   A_Number    : out Number) do
         …
      end loop;
   …
end Operator;
```

Invocazione

Realizzazione di accettazione

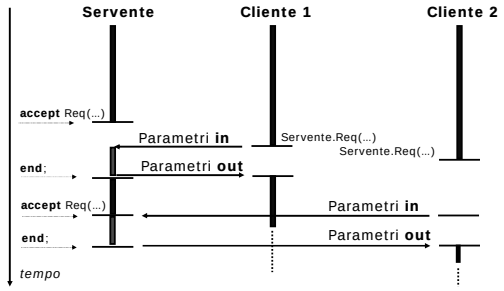
Modello base – 3

- **Storicamente chiamato *rendezvous* per rappresentare il fatto che cliente e servente si incontrano presso uno specifico punto d'accesso nello stesso istante temporale**
- **Al momento dell'incontro i parametri di modo `in` passano dal cliente al servente**
- **Il servente esegue la realizzazione del servizio come una normale procedura e poi restituisce i parametri di modo `out` al cliente**
- **A quel punto la sincronizzazione si interrompe e i processi riprendono la loro esecuzione concorrente**

Modello base – 4

- **Nella forma base**
  - **Il servente si sospende in attesa di una richiesta**
    - Come previsto per l'entità *Server* nel modello di concorrenza di lezione C01
  - **Il cliente si sospende fino alla disponibilità del servizio**
  - **La chiamata del cliente viene posta in una coda associata al servizio (*entry queue*)**
  - **L'ordine di accodamento è normalmente FIFO ma può essere configurato diversamente**
    - Per esempio su base prioritaria, con le conseguenze che questo può comportare in termini di *liveness*

Modello base – 5



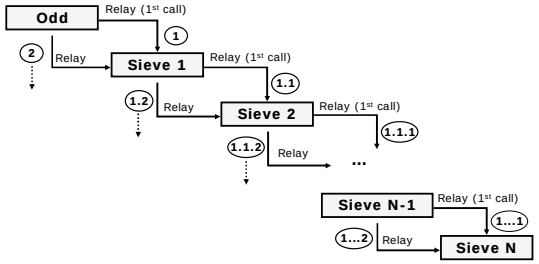
Un modello di *rendezvous*

Esempio – 1

- **Il crivello di Eratostene rivisitato**
  - **Una diversa realizzazione dell' algoritmo descritto nella lezione precedente**
  - **Invece di usare una risorsa protetta per scambiarsi i numeri da esaminare, ogni coppia di processi nella sequenza comunica tramite *rendezvous***
  - **L'effetto di sincronizzazione rende superflua la mutua esclusione, mentre l'accodamento FIFO sulla coda d'accesso preserva l'ordine dei valori da esaminare**

Un modello di *rendezvous*

Esempio – 2



Un modello di *rendezvous*

Sincronizzazione tripartita – 1

- **Il modello *rendezvous* è sincrono, asimmetrico e bidirezionale**
- **Le azioni svolte dal servente durante la sincronizzazione possono coinvolgere processi terzi**
  - **Ciò consente di realizzare forme complesse di sincronizzazione che però preservano separazione funzionale tra i processi coinvolti**
    - Proprietà desiderabile

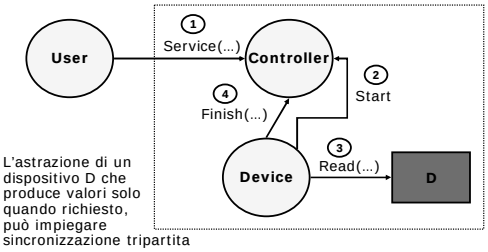
Un modello di *rendezvous*

Sincronizzazione tripartita – 2

- **Il coinvolgimento di processi terzi può avvenire in 2 forme strutturalmente distinte ma duali tra loro**
  - **Annidando accettazioni**
    - Modellando in tal modo una **macchina a stati** con alcuni stati raggiungibili solo a partire da un dato stato iniziale
  - **Invocando accettazioni nella realizzazione di una accettazione**
    - Realizzando così la fornitura di un **servizio composito** che racchiude il possibile contributo di più serventi che si siano ripartiti tra loro il lavoro

Un modello di *rendezvous*

Sincronizzazione tripartita – 3



L'astrazione di un dispositivo D che produce valori solo quando richiesto, può impiegare sincronizzazione tripartita

Un modello di *rendezvous*

Sincronizzazione tripartita – 4

```
task User;
task Device;
task Controller is
 entry Service (I : out Integer);
 entry Start;
 entry Finish (K : Integer);
end Controller;
```

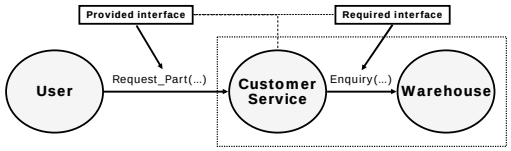
```
task body User is
…
 Controller.Service(Val);
 …
end User;
```

```
task body Controller is
begin
 loop
  accept Service (I : out Integer) do
   accept Start;
   accept Finish (K : Integer) do
    I := K;
   end Completed;
  end Service;
 end loop;
end Controller;
```

```
task body Device is
 Val : Integer;
 procedure Read (I : out Integer) is … ;
begin
 loop
  Controller.Start;
  Read(Val);
  Controller.Finish(Val);
end Device;
```

Un modello di *rendezvous*

Sincronizzazione tripartita – 5



Una fornitura di servizio che nasconda al cliente l'eventuale necessità di approvvigionamento da parte del servente presso componenti "nascosti" di una articolazione complessa di "sistema servente"

Strutturazione gerarchica con *information hiding*

Un modello di *rendezvous*

Sincronizzazione tripartita – 6

```
task Warehouse is
 entry Enquiry (Item : Part_Number;
                In_Stock : out Boolean);
end Warehouse;

task Customer_Service is
 entry Request_Part (Order : Part_Number;
                     Part  : Spare_Part;
                     Order : Order_Number);
end Customer_Service;
```

```
task body Customer_Service is
 In_Stock : Boolean;
 …
begin
 loop
  …
  accept Request_Part (Order : Part_Number;
                       Part  : Spare_Part;
                       Order : Order_Number) do
   …
   if In_Stock then
    Part  := The_Part;
    Order := None;
   else
    Warehouse.Enquiry(Order, In_Stock);
    if In_Stock then
     … -- go get the part from the Warehouse
     Part  := The_Part;
     Order := Next_Order_Number;
    end if;
   end if;
  end Request_Part;
 end loop;
end Customer_Service;
```


Un modello di *rendezvous*

Punti d'accesso privati – 1

- **Un processo servente non deve esporre necessariamente al pubblico tutti i suoi punti d'accesso**
- **Alcuni possono essere ristretti per motivi di convenienza (incapsulazione) e/o di opportunità (astrazione)**
- **La dichiarazione dei punti d'accesso deve in tal caso distinguere tra <u>pubblici</u> e <u>privati</u>**

Un modello di *rendezvous*

Punti d'accesso privati – 2

```
task User;
task Controller is
 entry Service (I : out Integer);
private
 entry Start;
 entry Finish (K : Integer);
end Controller;
```

La visibilità ai punti d'accesso privati è ristretta al solo ambito (*scope*) del processo Controller

```
task body User is
…
 Controller.Service(Val);
…
end User;
```

```
task body Controller is
 task Device;
 task body Device is
  Val : Integer;
  procedure Read (I : out Integer) is … ;
 begin
  loop
   Controller.Start;
   Read(Val);
   Controller.Finish(Val);
 end Device;
 -- continues in sidebox
```

```
begin -- Controller
 loop
  accept Service (I : out Integer) do
   accept Start;
   accept Finish (K : Integer) do
    I := K;
   end Completed;
  end Service;
 end loop;
end Controller;
```


Un modello di *rendezvous*

Casi d'errore

- **Un errore (eccezione) che occorra durante la sincronizzazione ne causa l'abbandono e si propaga a <u>entrambi</u> i partecipanti**
- **Emettere una richiesta d'accesso verso un processo terminato è considerato un errore a tempo di esecuzione e solleva una eccezione presso il chiamante**

Un modello di *rendezvous*

Stati d'esecuzione di processo