




## Comunicazione sincrona



Anno accademico 2019/2020  
Sistemi Concorrenti e Distribuiti

Tullio Vardanega, [tullio.vardanega@unipd.it](mailto:tullio.vardanega@unipd.it)

Laurea Magistrale in Informatica, Università di Padova
1/40




Comunicazione sincrona

## Modello base – 1

- ❑ **Stile di interazione cliente-servente**
  - Il servente specifica (espone) l'interfaccia dei servizi che offre
    - Tramite canali tipati di accesso (*entry*)
    - Ciascun canale con il proprio protocollo di scambio parametri
  - Il cliente fa richiesta (*entry call*) nominando il servente e il canale di interesse
  - Il servente fornisce il proprio servizio tramite accettazione esplicita della richiesta corrispondente
  - Il contatto tra cliente e servente è sincrono
    - L'erogazione di servizio non richiede necessariamente scambio di dati

Laurea Magistrale in Informatica, Università di Padova
2/18



Comunicazione sincrona

## 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;


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

Laurea Magistrale in Informatica, Università di Padova
3/40



Comunicazione sincrona

## Modello base – 3

- ❑ **Storicamente chiamato *rendez-vous***
  - Cliente e servente devono incontrarsi su uno specifico canale nello stesso istante temporale
- ❑ **Al momento dell'incontro, i parametri di modo in passano dal cliente al servente**
- ❑ **Il servente esegue il servizio richiesto come una normale procedura totalmente incapsulata in esso**
- ❑ **A fine lavoro restituisce i parametri di modo out al cliente sul canale**
- ❑ **A quel punto la sincronizzazione si scioglie e i due processi riprendono la loro esecuzione concorrente**

Laurea Magistrale in Informatica, Università di Padova
4/40



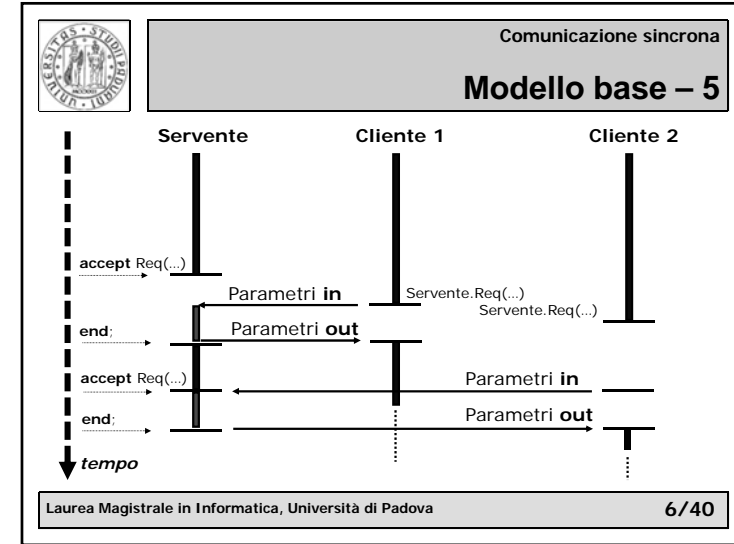
Comunicazione sincrona

## Modello base – 4

- ❑ Il server si sospende in attesa di richieste
- ❑ Il cliente si sospende in attesa del server
- ❑ La chiamata del cliente viene posta in una coda associata al canale (*entry queue*)
  - L'ordine base di accodamento è FIFO
  - Può essere configurato diversamente (p.es. base prioritaria), ma sotto rischio di *starvation*

Laurea Magistrale in Informatica, Università di Padova

5/18





Comunicazione sincrona

## Sincronizzazione tripartita – 1

- ❑ Il modello *rendez-vous* è
  - Sincrono rispetto alla comunicazione
  - Asimmetrico rispetto a interfaccia e denominazione
  - Bidirezionale rispetto al flusso dei dati
- ❑ Le azioni del server nella sincronizzazione possono coinvolgere processi terzi
  - Per realizzare forme avanzate di sincronizzazione preservando separazione funzionale tra le parti

Laurea Magistrale in Informatica, Università di Padova

7/40



Comunicazione sincrona

## Sincronizzazione tripartita – 2

- ❑ Il coinvolgimento di processi terzi nel lato *server* della sincronizzazione ammette due forme distinte e duali
  - Anidamento di accettazioni
    - Realizzando **macchine a stati** che usano accettazioni come *trigger*
  - Invocazione di richieste entro accettazioni
    - Realizzando un **servizio composito** che orchestra la ripartizione di lavoro di più server

Laurea Magistrale in Informatica, Università di Padova

8/40

Comunicazione sincrona

## Annidamento di accettazioni – 1

```

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

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

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

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

Laurea Magistrale in Informatica, Università di Padova **9/40**

Comunicazione sincrona

## Annidamento di accettazioni – 2

```

sequenceDiagram
    participant User
    participant Controller
    participant Device
    participant D

    User->>Controller: Service(...)
    activate Controller
    Controller->>Device: Start
    activate Device
    Device->>D: Read(...)
    activate D
    D-->>Device: 
    deactivate D
    Device-->>Controller: Finish(...)
    deactivate Device
    Controller-->>User: 
    deactivate Controller
    
```

- Il dispositivo D produce valori a richiesta
- Device realizza una macchina a stati i cui trigger provengono dall'accettazione di entry call
- Controller incapsula il protocollo d'uso di Device

Laurea Magistrale in Informatica, Università di Padova **10/40**

Comunicazione sincrona

## Servizio che incapsula richiesta – 1

```

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 part from Warehouse
      Part := The_Part; Order := Next_Order_Nr;
    end if;
  end if;
  end Request_Part;
  end loop;
end Customer_Service;
    
```

**Difetto:** in questo modo, i servizi di Warehouse sono visibili pubblicamente, non privati di Customer\_Service

Laurea Magistrale in Informatica, Università di Padova **11/40**

Comunicazione sincrona

## Servizio che incapsula richiesta – 2


```

sequenceDiagram
    participant User
    participant Customer_Service
    participant Warehouse

    User->>Customer_Service: Request_Part(...)
    activate Customer_Service
    Customer_Service-->>Warehouse: Enquiry(...)
    activate Warehouse
    Warehouse-->>Customer_Service: 
    deactivate Warehouse
    Customer_Service-->>User: 
    deactivate Customer_Service
    
```

- Strutturazione gerarchica con *encapsulation*
- Il servizio Request\_Part(...) nasconde al cliente la necessità di approvvigionamento presso fornitori terzi
- Questi ultimi, per trasparenza, dovrebbero essere incapsulati all'interno del servente

Laurea Magistrale in Informatica, Università di Padova **12/40**




Comunicazione sincrona

## Punti d'accesso privati – 1

- ❑ Un servente non deve necessariamente esporre pubblicamente tutti i suoi canali
- ❑ Alcuni di essi possono essere ristretti per motivi di incapsulazione e/o di astrazione
- ❑ In tal caso l'interfaccia esposto deve distinguere tra canali pubblici e privati

Laurea Magistrale in Informatica, Università di Padova

13/40



Comunicazione sincrona

## Punti d'accesso privati – 2

```

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

In questo modo la visibilità ai canali 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 loop;
    end Device;
  -- contains in sidebox
  begin -- Controller
    loop
      accept Service (I : out Integer) do
        accept Start;
        accept Finish (K : out Integer) do
          I := K;
        end Completed;
      end Service;
    end loop;
  end Controller;
        
```

Laurea Magistrale in Informatica, Università di Padova

14/40



Comunicazione sincrona

## Casi d'errore

- ❑ Una eccezione sollevata durante la sincronizzazione ne causa l'abbandono e si propaga a entrambi i partecipanti
- ❑ Emettere una richiesta d'accesso verso un processo terminato è un errore a tempo di esecuzione e solleva una eccezione nel chiamante

Laurea Magistrale in Informatica, Università di Padova

15/40




Comunicazione sincrona

## Limiti del modello base

- ❑ Il servente può accettare richieste su un solo canale alla volta
  - Il cliente può inviare una sola richiesta alla volta
- ❑ Una volta postosi in attesa su canale, il servente vi resta fino all'arrivo di una richiesta
  - Il cliente che invia richiesta resta sospeso in attesa della sua accettazione da parte del servente e del completamento delle azioni corrispondenti

Laurea Magistrale in Informatica, Università di Padova


16/40

 Comunicazione sincrona

## Requisiti di estensione – 1

- ❑ **Requisiti servente (RS) – più critici**
  1. **Poter attendere su più canali simultaneamente**
    - Comportamento naturale per un vero *server*
  2. **Limitare l'attesa a un tempo limite (*time-out*)**
  3. **Poter abbandonare immediatamente l'attesa su un canale che non abbia richieste in coda**
  4. **Poter terminare quando nessun cliente fosse più in grado di emettere richieste**
    - Comportamento desiderabile per un vero *server*

Laurea Magistrale in Informatica, Università di Padova 17/40

 Comunicazione sincrona

## Osservazione

- ❑ **I requisiti RS 1 e RS 3 sono assimilabili a quanto previsto dal modello "*Guarded Commands*" di Dijkstra**
- ❑ **I requisiti RS 2 e RS 4 hanno motivazione più pragmatica e meno algebrica**
  - Ma con conseguenze da considerare con cautela


Laurea Magistrale in Informatica, Università di Padova 18/40

 Comunicazione sincrona

## Requisiti di estensione – 2

- ❑ **Requisiti cliente (RC) – meno critici**
  - **Un singolo cliente sequenziale non può inviare più richieste alla volta (ovviamente)**
    - Per l'invio di più richieste simultanee servono clienti paralleli
  - 1. **Poter abbandonare l'attesa di accettazione ove essa non fosse disponibile immediatamente (cf. RS 2)**
  - 2. **Poter fissare un tempo limite all'attesa dell'accettazione di una richiesta (cf. RS 3)**

Laurea Magistrale in Informatica, Università di Padova 19/40

 Comunicazione sincrona

## Estensioni di lato servente – 1

- ❑ **RS 1. Attesa su più punti d'accesso**
  - **Il servente può erogare più servizi, ciascuno attraverso canale tipato dedicato (*entry*)**

```
task Server is
entry S1 (...);
entry S2 (...);
end Server;
```

```
task body Server is
...
begin
loop
select
accept S1(...) do ... end S1;
or
accept S2(...) do ... end S2;
end select;
end loop;
end Server;
```

Laurea Magistrale in Informatica, Università di Padova 20/40




Comunicazione sincrona

## Estensioni di lato serverente – 2

- ❑ RS 1. (continua)
  - Se nessuna richiesta fosse disponibile su alcun canale al momento della valutazione il serverente si pone in attesa
    - Sulla `select`
  - La valutazione avviene sempre simultaneamente su tutti i canali considerati
  - Qualora canali diversi avessero richieste in attesa, la scelta tra essi è non deterministica
    - Come nel modello di Dijkstra
  - La politica base per l'attesa su canale è FIFO
    - Altre politiche (p.es. per urgenza) possono essere contemplate

Laurea Magistrale in Informatica, Università di Padova

21/40



Comunicazione sincrona

## Estensioni di lato serverente – 3


- ❑ RS 1. (continua)
  - Opportuno aderire più pienamente al modello di Dijkstra
  - Porre "guardie" sui canali per specificare le condizioni logico-funzionali sotto le quali richieste su quel canale possano essere accettate

```
select
  Guard_1 => accept ...;
or
  Guard_2 => accept ...;
or
  ...
or
  Guard_N => accept ...;
end select;
```

- La guardia è una espressione Booleana, di tipo "when <condizione>" il cui verificarsi abilita la considerazione del canale
- Le guardie entro un comando `select` sono valutate simultaneamente e una sola volta all'inizio del comando

Laurea Magistrale in Informatica, Università di Padova

22/40




Comunicazione sincrona

## Estensioni di lato serverente – 4

- ❑ RS 2 e RS 3 hanno entrambi l'obiettivo di limitare superiormente il tempo di attesa, ma con semantiche diverse
  - RS 2. Abbandonare immediatamente l'attesa, in assenza di richieste all'istante di valutazione
    - Tempo di attesa nullo
  - RS 3. Fissare un tempo limite non nullo entro il quale il serverente è disposto ad attendere l'arrivo di richieste su uno dei canali considerati
    - Tempo di attesa relativo o assoluto, secondo bisogno

Laurea Magistrale in Informatica, Università di Padova

23/40



Comunicazione sincrona

## Estensioni di lato serverente – 5

- ❑ RS 2. Attesa nulla
  - Il serverente può voler considerare solo canali che abbiano richieste in attesa al momento del controllo altrimenti effettuare azioni alternative
    - Questo rende possibile l'attesa attiva anche se indesiderabile!

```
select
  accept A;
or
  accept B;
else
  C;
end select;
```

L'effetto richiesto può essere ottenuto in 2 modi alternativi


Forma esplicita (preferibile)

```
select
  accept A;
or
  accept B;
or
  delay T;
  C;
end select;
```

Forma implicita per T=0,0

Laurea Magistrale in Informatica, Università di Padova

24/40



Comunicazione sincrona


## Estensioni di lato servente – 6

❑ **RS 3. Attesa finita (> 0)**

- Un limite temporale di attesa non nullo consente al servente di adempiere al suo compito base
  - Senza però diventare incapace di fare altro a causa di attese infinite
- Al perdurare di assenza di richieste sui suoi canali il servente può assumere che i clienti si trovino in una condizione di errore
  - Senza che l'eventuale anomalia si propaghi al servente

Laurea Magistrale in Informatica, Università di Padova

25/40



Comunicazione sincrona

## Esempio – 1

```


task type Watchdog (Minimum_Distance : Duration) is
entry All_is_Well;
end Watchdog;

task body Watchdog is
begin
loop
select
accept All_is_Well;
... -- client is alive and well
or
delay Allowable_Distance;
... -- client may have failed, raise alarm
end select;
end loop;
end Watchdog;
    
```

Il modello delle guardie di Dijkstra applica anche all'alternativa di attesa temporale che può pertanto ammettere una guardia

Laurea Magistrale in Informatica, Università di Padova

26/40



Comunicazione sincrona

## Esempio – 2

```

with Ada.Real_Time; use Ada.Real_Time;
task Sensor_Monitor is
entry New_Period (Period : Time_Span);
end Sensor_Monitor;


...
task body Sensor_Monitor is
My_Period : Time_Span := Milliseconds(10_000);
Next_Cycle : Time := Clock + My_Period;
begin
loop
select
accept New_Period (Period : Time_Span) do
My_Period := Period;
Next_Cycle := Clock + My_Period;
delay until Next_Cycle;
or
delay until Next_Cycle;
-- do periodic work (e.g., read sensor)
Next_Cycle := Next_Cycle + My_Period;
end select;
end loop;
end Sensor_Monitor;
    
```

• Comportamento di base periodico, con lettura di sensore ogni 10 secondi  
 • Capace di variare dinamicamente l'ampiezza del periodo in caso di richiesta del cliente

L'effetto del cambiamento è ottenuto dalla presenza del blocco di comandi in ①

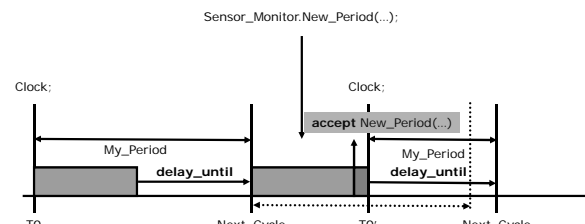
Laurea Magistrale in Informatica, Università di Padova

27/40



Comunicazione sincrona

## Esempio – 1: l'effetto



❑ L'arrivo di una richiesta "New\_Period" crea un nuovo riferimento T0' per i successivi periodi

- Interrompendo la periodicità precedente
- La soluzione data in ① comporta discontinuità temporale


Laurea Magistrale in Informatica, Università di Padova

28/40

Comunicazione sincrona

## RS 2 vs. RS 3

- ❑ Il *runtime* fa cose diverse nei due casi
- ❑ Per RS 3 deve «armare la sveglia»
  - A meno di valore costante fissato a 0 a tempo di compilazione
- ❑ Per RS 2 non deve farlo



Laurea Magistrale in Informatica, Università di Padova 29/40

Comunicazione sincrona

## Estensioni di lato servente – 7

- ❑ RS 4. Terminazione in mancanza di clienti
  - L'indipendenza tra clienti e servente può far sì che il servente sopravviva al completamento dei suoi clienti
    - In questo caso è desiderabile che anche il servente possa terminare
  - La terminazione del servente può essere gestita programmaticamente
    - Per esempio con valori sentinella, che avrebbero però impatto sulla natura architetturale dei serventi coinvolti
  - Trattandosi di un requisito generale di modello di *rendezvous* è bene disporre di una soluzione generale
    - Basta aggiungere un'alternativa `terminate` nel comando `select`

Laurea Magistrale in Informatica, Università di Padova 30/40

Comunicazione sincrona

## Estensioni di lato servente – 8


- ❑ RS 4. (continua)
  - Un servente sospeso su comando `select` con alternativa `terminate` aperta viene considerato “completo” allorché
    - Il master da cui esso dipende ha completato la propria esecuzione
    - E ogni altro processo dipendente da quello stesso master è:
      1. Già terminato
      2. Oppure sospeso su un comando `select` con alternativa `terminate` aperta
  - La condizione 1 assicura che non vi possano essere nuove richieste di servizio in arrivo
  - La condizione 2 applica transitivamente e quindi segnala un sistema totalmente inattivo

Laurea Magistrale in Informatica, Università di Padova 31/40

Comunicazione sincrona


## Ultime volontà 😊

- ❑ La semantica di terminazione RS 4 va arricchita in modo da permettere al processo terminante di effettuare azioni esplicite di finalizzazione
  - Le ultime volontà ...
- ❑ Alcuni tipi esportano un metodo di finalizzazione che viene invocato dal *runtime* quando un oggetto di quel tipo esce di *scope*
  - La terminazione di un processo il cui *scope* contenga istanze di tali tipi comporta l'invocazione automatica dei loro metodi di finalizzazione



Laurea Magistrale in Informatica, Università di Padova 32/40





Comunicazione sincrona


## Esempio – 3

❑ **Il crivello di Eratostene sincrono**

- Ogni coppia di processi annidati nella sequenza algoritmica comunica tramite *rendez-vous*
- L'effetto di sincronizzazione rende superflua la mutua esclusione sui dati del servizio
  - L'accodamento FIFO sulla coda d'accesso preserva l'ordine dei valori da esaminare (proprietà di serializzazione)
  - Nell'esempio, i dati passano in modo *in*, quindi in sola lettura
- Aggiungiamo controllo di terminazione alle istanze dei processi «crivello»
  - In caso di terminazione, vogliamo che ogni processo terminante ce ne fornisca notifica

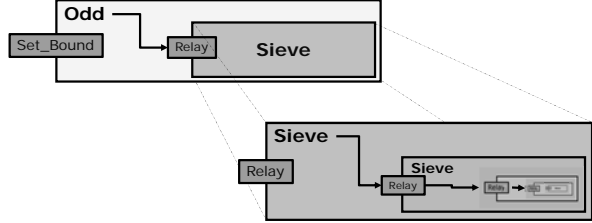
Laurea Magistrale in Informatica, Università di Padova

33/40



Comunicazione sincrona

## Esempio – 4



❑ La natura ricorsiva dell'algoritmo traspare nell'annidamento gerarchico dei processi (Odd (Sieve)) e (Sieve (Sieve))

- La profondità di ricorsione ignota a priori del problema comporta creazione dinamica di processi risolutori
- Questa non è una buona prassi!

Laurea Magistrale in Informatica, Università di Padova

34/40



Comunicazione sincrona


## Estensioni di lato cliente

❑ Per il lato cliente avevamo solo 2 esigenze

- RC 2. Abbandonare l'attesa per servente non immediatamente disponibile
  - Equivalente al requisito RS 2 del lato servente
- RC 3. Limite temporale non nullo all'attesa di servizio
  - Equivalente al requisito RS 3 di lato servente
  - Il limite riguarda solo la durata massima di attesa fino a inizio sincronizzazione
  - Nessuna relazione con la durata effettiva della sincronizzazione!

Laurea Magistrale in Informatica, Università di Padova

35/40



Comunicazione sincrona

## Usi del modello cliente-servente

❑ Un servente è un'entità reattiva capace di garantire mutua esclusione

- Eseguendo una sola alternativa accept alla volta

❑ L'esecuzione della sincronizzazione rappresenta una sezione critica

❑ La risorsa condivisa deve però essere visibile soltanto al processo servente

```

task body Buffer (...) is
  ... -- the shared resource
begin
  task type Buffer (...) is
  ...
  loop
  select
  when ...
  accept Put (...) do ... end Put;
  ... -- local housekeeping
  or
  when ...
  accept Get (...) do ... end Get;
  ... -- local housekeeping
  or
  terminate;
  end select;
end loop;
end Buffer;
    
```

Laurea Magistrale in Informatica, Università di Padova

36/40

Comunicazione sincrona

### Abusi del modello cliente-servente

- ❑ Programmazione incauta può causare situazioni di stallo
  - Anche il modello *rendez-vous* esteso non è capace di impedirne strutturalmente il rischio

```

task T1 is
entry A;
end T1;

...

task body T1 is
begin
T2.B;
accept A;
end T1;
            
```

```

task T2 is
entry B;
end T2;

...

task body T2 is
begin
T1.A;
accept B;
end T1;
            
```

Laurea Magistrale in Informatica, Università di Padova 37/40

Comunicazione sincrona

### Una buona prassi

- ❑ I processi dovrebbero essere usati soltanto per realizzare entità attive oppure *server*
  - Secondo una ben precisa architettura di sistema
  - Trattando ogni dipendenza esterna tramite incapsulazione
- ❑ Le entità con ruolo attivo non dovrebbero possedere canali ma solo inviarvi messaggi
- ❑ I *server* "puri" accettano richieste di accesso ma non ne fanno alcuna

Laurea Magistrale in Informatica, Università di Padova 38/40

Comunicazione sincrona

### Stati d'esecuzione di processo

Laurea Magistrale in Informatica, Università di Padova 39/40

Comunicazione sincrona

### Stati d'esecuzione di processo

Laurea Magistrale in Informatica, Università di Padova 40/40