Estensioni del modello *rendez-vous*

SCD

Anno accademico 2015/16
Sistemi Concorrenti e Distribuiti

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

Limiti del modello base

- Nel modello di concorrenza noto finora il servente può accettare richieste su un solo canale alla volta
  - Analogamente il cliente può inviare una sola richiesta alla volta
- Una volta in attesa sul canale il servente attende fino all'arrivo di una richiesta
  - Inviata la richiesta, il cliente resta sospeso in attesa della sua accettazione e del completamento delle relative azioni

Requisiti di estensione – 1

- Requisiti servente (RS) – più critici
  1. Poter attendere su più di un canale alla volta
  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
     - Il comportamento più naturale per un vero *server*

Osservazione

- I requisiti RS 1 e RS 3 sono assimilabili a quanto previsto dal modello "*Guarded Commands*" di Dijkstra
  - Lezione C02
- I requisiti RS 2 e RS 4 hanno invece una motivazione meno algebrica e più pragmatica
  - Ma con conseguenze da considerare con cautela

Requisiti di estensione – 2

- **Requisiti cliente (RC) – meno critici**
  - **A un singolo cliente non serve poter inviare più richieste alla volta**
    - Un processo cliente funzionalmente coeso ha una logica interna sequenziale
    - Se serve inviare più richieste simultaneamente servono più clienti paralleli
  2. **Può servire fissare un tempo limite all'attesa dell'accettazione di una richiesta (cf. RS 2)**
  3. **È utile poter abbandonare l'attesa di accettazione qualora questa non fosse immediatamente disponibile (cf. RS 3)**

Estensioni di lato servente – 1

- **RS 1. Attesa su più punti d'accesso**
  - **Il servente può erogare più servizi, ciascuno di essi fornito attraverso messaggi scambiati su un 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;
```


Estensioni di lato servente – 2

- **RS 1. (continua)**
  - **Se nessuna richiesta fosse disponibile su alcun canale al momento della valutazione il servente 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

Estensioni di lato servente – 3

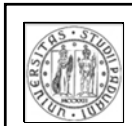
- **RS 1. (continua)**
  - **Opportuno aderire più pienamente al modello di Dijkstra**
  - **Porre "guardie" sui canali per specificare le condizioni logico-funzionali sotto le quali richieste su quel canale possano essere accettate**

```
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

Estensioni di lato servente – 4

- **RS 2 e RS 3 hanno entrambi l'obiettivo di limitare il tempo massimo di attesa ma con semantiche diverse**
  - **RS 2. Fissare un tempo limite non nullo entro il quale il servente è disposto ad attendere l'arrivo di richieste su uno dei canali considerati**
    - Tempo di attesa relativo o assoluto a seconda della necessità applicative
  - **RS 3. Abbandonare immediatamente l'attesa in assenza di richieste all'instante di valutazione**
    - Equivalente ad ammettere solo tempo di attesa nullo



Estensioni di lato servente – 5

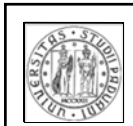
- **RS 2.**
  - **Un limite temporale di attesa non nullo consente al servente di adempiere al suo compito base**
    - Senza però diventare incapace di fare altro a causa di attese infinite
  - **Al perdurare di assenza di richieste sui suoi canali il servente può assumere che i clienti si trovino in una condizione di errore**
    - Ma con questa modalità l'eventuale anomalia non si propaga al servente

Esempio – 1

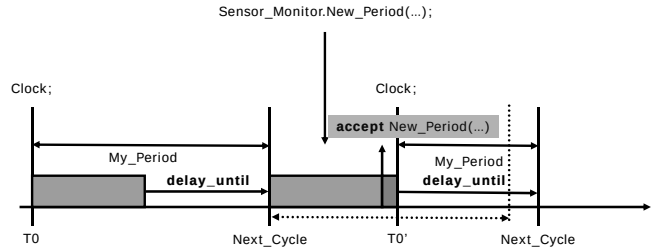
```
with Ada.Real_Time; use Ada.Real_Time;
task Sensor_Monitor is
 entry New_Period (Period : Time_Span);
end Sensor_Monitor;
…
task body Sensor_Monitor is
 My_Period : Time_Span := Milliseconds(10_000);
 Next_Cycle : Time := Clock + My_Period;
begin
 loop
  select
   accept New_Period (Period : Time_Span) do
    My_Period := Period;
   end New_Period;
   Next_Cycle := Clock + My_Period;
   delay until Next_Cycle;                    ①
  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



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**

Esempio – 2

```
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

Estensioni di lato servente – 6

- **RS 3. Attesa nulla**
  - **Il servente può voler prestare attenzione solo a quei canali che abbiano già richieste in attesa <u>al momento del controllo</u> 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

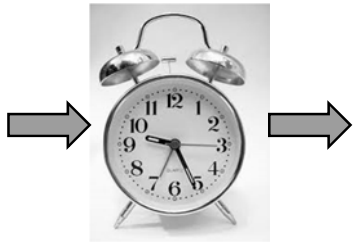
Forma esplicita (preferibile)

Forma implicita per T=0.0

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


RS 2 vs. RS 3

- **La macchina astratta fa cose diverse nei due casi**
- **Per RS 2 deve «armare la sveglia»**
  - **A meno di valore costante fissato a 0 a tempo di compilazione**
- **Per RS 3 non ne ha bisogno**



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 nella versione bis del crivello di Eratostene
      - Ma in quel programma non agirebbero serventi "puri" ma degeneri!
  - **Trattandosi però di un requisito generale di modello di *rendez-vous* è bene disporre di una soluzione generale**
    - Basta aggiungere un'alternativa **`terminate`** nel comando **`select`**

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
    - Ogni altro processo dipendente da quello stesso master è
      - Già terminato, oppure
      - A sua volta 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**

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 dalla macchina astratta 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**

Esempio – 3

- **Il crivello di Eratostene (versione Ter)**
  - **Vogliamo aggiungere controllo di terminazione alle istanze dei processi di tipo Sieve_T**
  - **In caso di terminazione, vogliamo anche che il processo terminante ce ne fornisca notifica**

Estensioni di lato cliente

- **Per il lato cliente avevamo solo 2 esigenze**
  - **RC 2. Porre un limite temporale non nullo all'attesa di servizio**
    - Equivalente al requisito **RS 2** di lato servente e soddisfatto nello stesso modo
    - Il limite riguarda solo la durata massima dell'attesa fino all'inizio della sincronizzazione
    - Nessuna relazione con la durata effettiva della sincronizzazione!
  - **RC 3. Lasciare l'attesa immediatamente qualora il servente non fosse subito disponibile**
    - Equivalente al requisito **RS 3** del lato servente e soddisfatto analogamente
    - Sta alla realizzazione del modello trattare il caso in cui più clienti desiderino simultaneamente conoscere la disponibilità istantanea di uno stesso servente

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 la sezione critica**
- **La risorsa condivisa deve però essere visibile soltanto al processo servente**

```
task body Buffer (…) is
 … -- the shared resource
begin
 …
 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;
```

```
task type Buffer (…) is
 entry Put (…);
 entry Get (…);
end Buffer;
```


Abusi del modello cliente-servente

- **Programmazione incauta può causare situazioni di stallo**
  - **Anche nella sua forma estesa il modello *rendez-vous* 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;
```


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**

Stati d'esecuzione di processo