

# Progettazione di un sistema Concorrente e Distribuito

Moreno Ambrosin

Università degli studi di Padova

Dipartimento di Matematica

Corso di laurea Magistrale in Informatica

Settembre 2013

# Indice

- 1 Introduzione
- 2 Analisi del Problema
- 3 Costruzione di una soluzione
- 4 Scelta degli strumenti tecnologici
- 5 Conclusioni

# Introduzione

- La progettazione di un sistema concorrente e distribuito si compone di:
  - Analisi del problema.
    - Definizione dei requisiti del sistema.
    - Analisi degli aspetti legati alla distribuzione.
    - Analisi degli aspetti legati alla concorrenza.
  - Costruzione di una soluzione.
    - Definizione di architettura di distribuzione delle componenti.
    - Definizione del protocollo di distribuzione.
    - Risoluzione delle problematiche di concorrenza locali alle componenti del sistema.
- Scelta del supporto tecnologico da utilizzare per l'implementazione.

# Analisi del Problema

- Individuazione e prima definizione delle entità del sistema.
  - ad es. il progetto di un simulatore per un sistema ferroviario comprende:
    - Treno
    - Viaggiatore
    - Segmento
    - Stazione
      - Piattaforma
      - Biglietteria
      - Pannello Informativo
    - Controllo Centrale
- Identificazione e definizione dei requisiti del sistema.

# Analisi del Problema - Distribuzione (1)

- Primo aspetto da valutare.
  - Le scelte a livello di distribuzione influenzano
    - la natura delle entità;
    - le modalità di interazione tra di esse.
- Scelte architetturali:
  - prima modellazione delle componenti (cosa distribuire);
  - dove adottare distribuzione *verticale* o *orizzontale*;
  - modalità di comunicazione tra le componenti (*sincrona* o *asincrona*);
  - definizione di possibili interfacce.

# Analisi del Problema - Distribuzione - Caratteristiche desiderabili (1)

- Il sistema dovrà apparire agli utenti come unitario e coerente.
- **Trasparenza:**
  - Il sistema dovrà il più possibile rendere trasparenti all'utente le caratteristiche legate alla distribuzione (Accesso, Collocazione, Migrazione, Spostamento, Replicazione, Malfunzionamento, Persistenza)
- **Openess:**
  - Il sistema dovrà garantire portabilità e interoperabilità.
  - Il sistema dovrà essere fruibile mediante regole standard (interfacce).
  - Organizzazione del sistema in componenti di dimensione ridotta, e facilmente sostituibili.
  - Separazione tra *politiche* e *meccanismi*.

## Analisi del Problema - Distribuzione - Caratteristiche desiderabili (2)

### ■ Scalabilità:

- Rispetto alla cardinalità del sistema (ad es. nel progetto di un sistema ferroviario, è desiderabile poter aumentare la popolazione di Stazioni e Segmenti di collegamento).
- Rispetto alla distribuzione spaziale delle componenti.
- Rispetto alle problematiche locali di gestione (che non devono affliggere l'intero sistema).

### ■ Fault Tolerance:

- Il sistema deve essere progettato in modo tale da ridurre l'impatto causato da *partial failures*.
- Il sistema dovrà gestire errori di comunicazione tra i nodi.

### ■ Avvio ordinato

### ■ Terminazione in uno stato consistente

## Analisi del Problema - Distribuzione (2)

- Valutazione di diverse possibili architetture di sistema.
  - Possibile introduzione di nuove entità nel problema
    - ad es.: introduzione dell'entità Regione, raggruppamento di Stazioni, Segmenti di collegamento, Treni e Viaggiatori su un singolo nodo di calcolo.
  - Adozione di gradi di distribuzione diversi
    - come influiscono sulla definizione del protocollo di distribuzione;
    - quali sono le conseguenze sul sistema (ad es., su realismo della simulazione e consistenza temporale).



# Analisi del Problema - Distribuzione - Esempio (1)

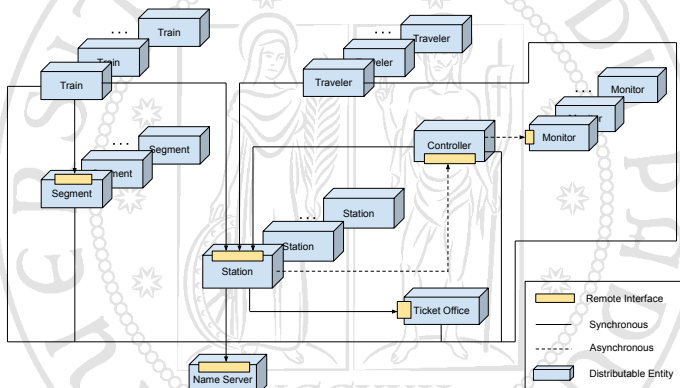


Figura: Architettura di alto livello in cui tutte le entità di simulazione principali sono distribuite. Per semplicità, interfaccia remota e comunicazione da e verso entità presenti con molteplicità maggiore di 1, sono indicate per una singola istanza.

## Analisi del Problema - Distribuzione - Esempio (2)

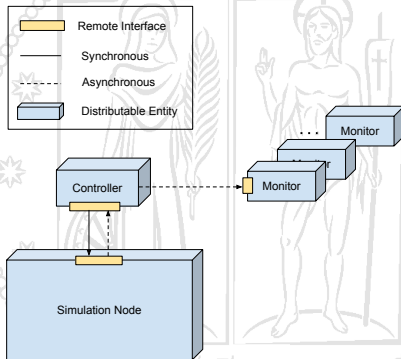


Figura: Architettura di alto livello in cui solo Controller Centrale e componente di Visualizzazione sono distribuite. Per semplicità, interfaccia remota e comunicazione da e verso entità presenti con molteplicità maggiore di 1, sono indicate per una singola istanza.

# Analisi del Problema - Distribuzione - Esempio (3)

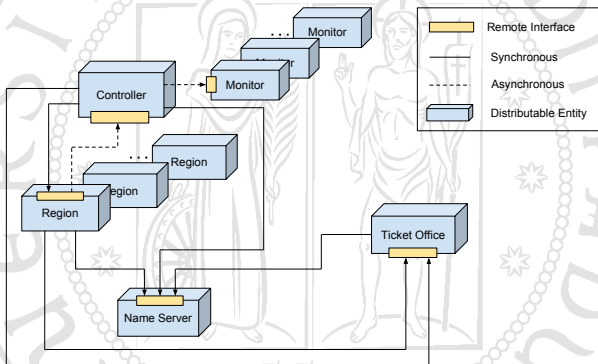


Figura: Architettura di alto livello con distribuzione a livello di *Regioni*. Soluzione adottata nel progetto didattico. Per semplicità, interfaccia remota e comunicazione da e verso entità presenti con molteplicità maggiore di 1, sono indicate per una singola istanza.

# Analisi del Problema - Concorrenza

- Prima definizione dei protocolli logici di interazione concorrente tra le entità del problema.
  - Indipendente dalla scelta di un modello di concorrenza specifico.
  - Identificazione dei punti critici in cui il problema esprime concorrenza.

# Analisi del Problema - Concorrenza - Esempio (1)

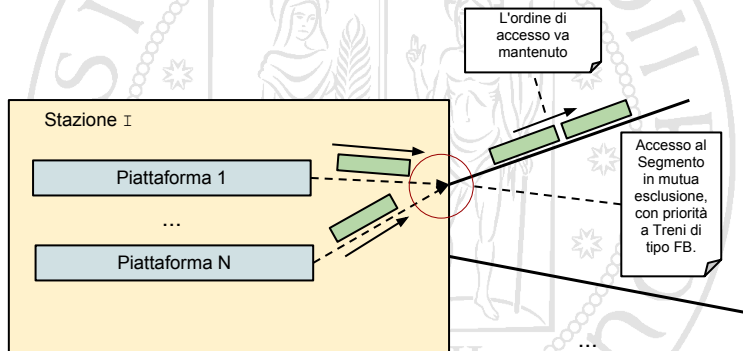


Figura: Accesso ad un segmento.

# Analisi del Problema - Concorrenza - Esempio (2)

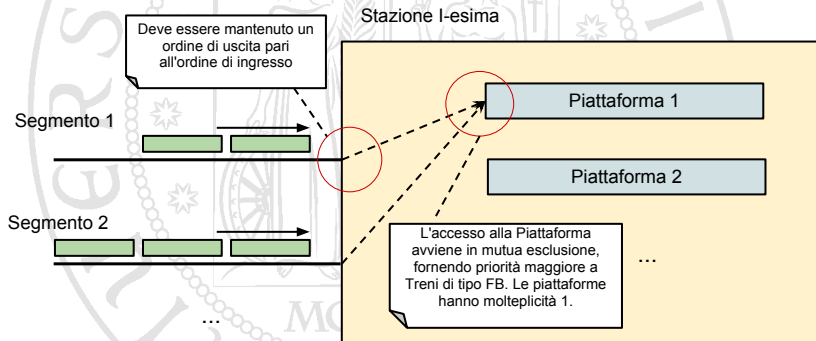


Figura: Uscita da un segmento e accesso alla Piattaforma successiva.

## Costruzione di una soluzione - Distribuzione

- Scelta dell'architettura di distribuzione da utilizzare, che meglio si adatta ai requisiti del problema.
  - ad es. Distribuzione a livello di Regioni permette di contenere il grado di distribuzione e garantisce alcune delle caratteristiche desiderabili.
- Definizione architetturale a grana più fine
  - ad es.: introduzione di una gerarchia di Biglietterie per distribuire conoscenza e oneri di calcolo.
    - Centrale
    - Regionale
    - Interna alle Stazioni
- Definizione del protocollo di distribuzione che realizza l'interazione tra le componenti, in base all'architettura scelta, e ai requisiti del problema.

## Costruzione di una soluzione - Distribuzione - Esempio

- Come realizzare il passaggio di entità Treno e Viaggiatore tra Regioni?
  - *possibile soluzione*: Utilizzare Stazioni speciali (di “gateway”) per permettere l’uscita di un Treno da una Regione; trasferimento diretto di un Viaggiatore.
- Come realizzare il trasferimento remoto di una entità?
  - creazione/distruzione?
  - replicazione?
  - *possibile soluzione*: Disaccoppiamento tra entità e thread che ne esegue le operazioni: utilizzo di pool di thread esecutori (*Train\_Executor* o *Traveler\_Executor*) e di descrittori di entità (*Train\_Descriptor* o *Traveler\_Descriptor*), quest’ultimi replicati su ciascun nodo e aggiornati al trasferimento.



## Costruzione di una soluzione - Distribuzione - Avvio

- L'avvio deve essere coordinato tra le componenti distribuite.
  - Devono essere evitati tentativi di comunicazione tra componenti non ancora pronte o allocate.
- È opportuno scegliere un ordine di avvio tra le componenti e separare la fase di inizializzazione delle componenti dall'avvio del sistema.

## Costruzione di una soluzione - Distribuzione - Terminazione

- La Terminazione ha come prerequisito la definizione dei limiti entro i quali uno stato del sistema è consistente.
- Ad es. nel progetto di un sistema ferroviario:
  - è accettabile che il sistema termini con un certo numero di Treni in attesa di accedere ad una Piattaforma;
  - non è accettato lo stato di terminazione per il quale un Viaggiatore è in attesa di un Biglietto.
- È conveniente adattare algoritmi distribuiti noti (ad es. *distributed snapshot*).
- Nessun thread in esecuzione sui nodi di calcolo dopo la procedura.

## Costruzione di una soluzione - Distribuzione - Valutazione (1)

- Architettura e protocollo di distribuzione vanno valutati sulla base delle caratteristiche desiderabili per un sistema distribuito, individuando criticità e punti di forza della soluzione.
- ad es. alcune caratteristiche della soluzione adottata:
  - Il sistema è scalabile in dimensione in relazione al numero di Regioni
    - La scalabilità del sistema è favorita dalla suddivisione a livelli delle Biglietterie e dal trasferimento di descrittori per il passaggio tra Regioni di Treni e Viaggiatori.
    - I pool di Train\_Executor e Traveler\_Executor di ciascun nodo devono essere dimensionati in modo opportuno per poter permettere scalabilità del sistema.

## Costruzione di una soluzione - Distribuzione - Valutazione (2)

- La natura distribuita del sistema è nascosta alla componente di Visualizzazione.
- Fallimenti a livello di singole Regioni possono essere tollerati dal sistema.
- Controllo Centrale e Server dei Nomi sono fattori di centralizzazione per il sistema
  - Protocolli di avvio e terminazione semplici, ma
  - Single Points of Failure.
  - Potenziali colli di bottiglia.
- In assenza di riferimento temporale assoluto, c'è la possibilità di inconsistenze temporali che devono essere corrette o evitate.

## Costruzione di una soluzione - Concorrenza (1)

- Risoluzione delle problematiche di concorrenza locali a ciascuna componente.
- Definizione e classificazione delle entità concorrenti
  - ad es. Segmento *entità reattiva* con agente di controllo, a molteplicità  $1 \leq n \leq N$ .
  - ad es. Train\_Executor e Traveler\_Executor sono *entità attive*, mentre Train\_Descriptor e Traveler\_Descriptor sono strutture dati semplici (ad es. record);  
⇒ La natura delle entità concorrenti dipende dal protocollo di interazione tra le componenti distribuite.

## Costruzione di una soluzione - Concorrenza (2)

- Scelta di un modello di concorrenza adatto alle caratteristiche del problema.
  - Valutazione di modelli differenti, ad es. modello ad *attori* o a *monitor*.
- Modellazione delle entità concorrenti e della loro interazione, con strumenti di modello.
  - Scomposizione in sottoproblemi semplici.
- Evitare scelte di progettazione che utilizzano operazioni specifiche offerte dalle tecnologie
  - Nessuna assunzione a priori sul linguaggio che verrà utilizzato.
  - Nessuna assunzione sulle politiche di scheduling adottate dalla macchina sottostante.

## Costruzione di una soluzione - Concorrenza - Esempio (1)

- Accesso ad un Segmento  $S$  da parte di un Train\_Executor  $T$ , che esegue per uno specifico Treno, dall'estremo  $D_T$ ,  $D_T = \text{First End}$ .
  - Rischio starvation di Train\_Executor in attesa di accedere.
- Segmento realizzato come risorsa protetta con agente di controllo *monitor*.  $T$  accede ad  $S$  sse:
  - $S$  è libero
  - $S$  non è libero ma i Train\_Executor "in transito" hanno avuto accesso da  $D_T$  e il numero di accessi massimo non è stato raggiunto.
  - $S$  non è libero ma i Train\_Executor "in transito" hanno avuto accesso da  $D_T$ , il numero di accessi massimo è stato raggiunto, ma all'estremo opposto non vi sono altri Train\_Executor in attesa.
- In tutti gli altri casi  $T$  deve attendere presso l'estremo di accesso.
- L'ultimo Train\_Executor che esce da  $S$  risveglia i Train\_Executor in attesa presso l'estremo opposto.

# Costruzione di una soluzione - Concorrenza - Esempio (2)

```
procedure Access_Segment_Monitor(T:Train,Access_End:Integer) begin
```

```
...
```

```
if Access_End = First_End then
```

```
while
```

```
((not Free) and (Access_End /= Current_Direction))
```

```
or
```

```
((not Free) and (Access_End = Current_Direction) and  
(Access_Number = MAX) and (Second_End_Count > 0))
```

```
loop
```

```
First_End_Count := First_End_Count + 1;
```

```
wait(Can_Enter_First_End);
```

```
First_End_Count := First_End_Count - 1;
```

```
end loop;
```

```
else
```

```
... // Simmetrico per accesso dalla direzione opposta
```

```
end if;
```

```
if (Free = True) then
```

```
Free := False;
```

```
if (Access_End /= Current_Direction) then
```

```
Access_Number := 1;
```

```
Current_Direction := Access_End;
```

```
end if;
```

```
else
```

```
if (Access_Number < MAX) then
```

```
Access_Number := Access_Number + 1;
```

```
end if;
```

```
end if;
```

```
...
```

```
end;
```

- Procedura che regola l'accesso ad un Segmento da parte di un Train\_Executor;
- Accesso multiplo al Segmento, con numero massimo *MAX* di ingressi consecutivi per estremo.
- Current\_Direction mantiene la direzione dei Train\_Executor "in transito".



## Costruzione di una soluzione - Concorrenza - Esempio (3)

Valutazione dei casi possibili per dimostrare la correttezza della soluzione presentata, una volta che  $T$  esegue all'interno della procedure di risorsa protetta `Access_Segment_Monitor`.

### Caso 1: Accesso Consentito

*Precondizione:* `Free=True`

$T$  imposta il valore di `Free` a `False`. Se l'estremo di accesso è diverso da quello corrente, allora il numero di accessi per estremo `Access_Number` viene incrementato di 1, e la direzione corrente `Current_Direction` è settata a 1. In questo modo una volta raggiunto il massimo numero di accessi `MAX`, esso viene re-impostato ad 1 solo se  $T$  proviene da una direzione diversa dall'ultima percorsa. Esegue infine le operazioni previste dopo aver ottenuto l'accesso.

## Costruzione di una soluzione - Concorrenza - Esempio (4)

### Caso 2: Accesso Consentito

*Precondizione:*  $\text{Free} = \text{False}$  and  $\text{Current\_Direction} = D_T$   
and  $1 < \text{Access\_Number} < \text{MAX}$

Perché sia verificata la Precondizione, almeno un altro *Train\_Executor* proveniente dallo stesso estremo deve aver avuto accesso ad *S* (Caso 1). In questo caso, *T* si limita a incrementare di 1 il contatore di accessi per estremo *Access\_Number*, e ad eseguire le operazioni previste dopo l'accesso.

### Caso 3: Accesso Consentito

*Precondizione:*  $\text{Free} = \text{False}$  and  $\text{Current\_Direction} = D_T$  and  
 $\text{Access\_Number} = \text{MAX}$  and  $\text{Second\_End\_Count} = 0$

Perché sia verificata la Precondizione, almeno *MAX* *Train\_Executor* provenienti dallo stesso estremo hanno avuto accesso ad *S* (Caso 1 + Caso 2). In questo caso, *T* non incrementare il contatore di accessi *Access\_Number*, ed esegue le operazioni previste dopo l'accesso.

## Costruzione di una soluzione - Concorrenza - Esempio (5)

### Caso 4: Accesso non consentito

*Precondizione:*  $\text{Free} = \text{False}$  and  $\text{Current\_Direction} \neq D_T$

Perché la pre-condizione sia verificata, almeno un `Train_Executor` proveniente dall'estremo opposto rispetto a  $T$  ha eseguito all'interno di `Access_Segment_Monitor` nel Caso 1. Il thread corrente incrementa il contatore dei `Train_Executor` in attesa per l'estremo corrente `First_End_Count`, e si pone in attesa su variabile di condizione `Can_Enter_First_End`.

### Caso 5: Accesso non consentito

*Precondizione:*  $\text{Free} = \text{False}$  and  $\text{Current\_Direction} = D_T$  and  $\text{Access\_Number} = \text{MAX}$  and  $\text{Second\_End\_Count} > 0$

Perché la pre-condizione sia verificata, almeno un `Train_Executor` proveniente dall'estremo opposto rispetto a  $T$  ha eseguito all'interno di `Access_Segment_Monitor` nel Caso 4 (relativamente alla propria direzione). Il thread corrente incrementa il contatore dei `Train_Executor` in attesa per l'estremo corrente `First_End_Count`, e si pone in attesa su variabile di condizione `Can_Enter_First_End`.

## Scelta degli strumenti tecnologici

- Utilizzo di linguaggi di programmazione e strumenti che meglio si adattano alle scelte di progetto .
- Interessante l'utilizzo di tecnologie eterogenee
  - È difficile pensare ad un sistema distribuito realizzato con tecnologia uniforme.
  - Possibilità di utilizzare supporti tecnologici specifici per singola componente.  
Ad es. nella soluzione progettata:
    - ho utilizzato il linguaggio Ada per codificare le componenti che rappresentano le Regioni;
    - ho utilizzato il linguaggio Scala per la realizzazione di Name Server, Biglietteria e Controller Centrale;
    - ho utilizzato il middleware a scambio di messaggi Yami4 per integrare componenti eterogenee.

# Conclusioni

- La progettazione di un sistema concorrente e distribuito è un processo complesso.
- Molto importante l'analisi iniziale, il confronto tra diverse possibili architetture di distribuzione e la validazione delle soluzioni adottate.
- Errori comuni:
  - dare per scontato problematiche di distribuzione;
  - affidarsi a strumenti di linguaggio per risolvere problemi di concorrenza;
  - progettazione della soluzione a partire dalle tecnologie.