Railway Simulation

Moreno Ambrosin
mat. 1035635

Progetto Sistemi Concorrenti e Distribuiti

Corso di laurea magistrale in Informatica
Università degli Studi di Padova
Padova

31 marzo 2013

Indice

Capitolo 1

Il problema

Il progetto didattico prevede la progettazione e la realizzazione di un simulatore software di un sistema ferroviario. Tale sistema prevede i seguenti requisiti di base (la specifica originale è consultabile all'indirizzo `http://www.math.unipd.it/~tullio/SCD/2005/Progetto.html`):

- La presenza di Treni appartenenti a categorie a priorità e modalità di fruizione diverse.
- La presenza di Viaggiatori, che eseguono operazioni elementari come acquisto di un biglietto, salita/discesa su/da un Treno, ecc.
- La definizione di un ambiente costituito da un insieme di Stazioni, ciascuna composta da:
  - Piattaforme di attesa per Treni e Viaggiatori.
  - Un Pannello Informativo che visualizza informazioni sui Treni in arrivo, in transito e che hanno appena superato la Stazione corrente.
  - Una Biglietteria, presso la quale un Viaggiatore può acquistare un biglietto di viaggio.
- La presenza di Segmenti di collegamento tra Stazioni, a percorrenza bidirezionale.
- La presenza di un entità di controllo globale che mantiene lo stato di ciascun Viaggiatore e ciascun Treno in transito.
- L'obbligo da parte di un Viaggiatore di acquistare un Biglietto prima di poter usufruire del servizio ferroviario.

- La presenza di collegamenti multipli tra Stazioni, ovvero ciascuna Stazione può essere raggiunta da più Segmenti e da ciascuna stazione possono partire più segmenti.
- Il percorso portato a termine da un Viaggiatore può comprendere cambi di treno.

1.1 Specifica arricchita

La soluzione presentata con il seguente documento, è relativa alla seguente raffinamento della specifica originaria:

- Ciascun Treno appartiene ad una delle seguenti due categorie:

  Regionale
  Treno a bassa priorità, senza posto garantito.

  FB
  Treno a priorità più alta, che necessita prenotazione.

- Ciascun Treno possiede una capienza massima di Viaggiatori.

Capitolo 2

Analisi del problema

La progettazione di un simulatore di un sistema ferroviario presenta diverse problematiche relativamente alla concorrenza e alla distribuzione, in quanto:

- il problema prevede l'interazione tra una popolazione di entità in maniera concorrente;
- vi sono dei punti di sicronizzazione tra le entità, che devono essere identificati e modellati opportunamente;
- non è ragionevole fare assunzioni a priori sulle tecnologie che risolveranno il problema, ne sull'ambiente di esecuzione;
- è ragionevole prevedere un certo livello di distribuzione delle componenti del sistema;
- si possono presentare difficoltà dovute ai ritradi nella trasmissioni di rete tra le componenti.

Di seguito andrò ad analizzare quelle che sono le principali.

2.1 Distribuzione

Le problematiche legate alla distribuzione sono molteplici. Nel progetto di un simulatore di un sistema ferroviario infatti, la presenza di entità distribuite è auspicabile, sia per suddividere logicamente le entità, sia per distribuire l'onere di calcolo su nodi differenti. Le caratteristiche desiderabili da un sistema dostribuito che simula una struttura ferroviaria sono:

- Il complesso deve apparire all'utente come un sistema unitario, la natura distibuita del sistema deve essere nascosta all'utilizzatore finale.

- L'architettura distribuita non deve limitare le funzionalità desiderate.
- É desiderabile che vi sia un buon grado di disaccoppiamento tra le componenti, e dalle tecnologie adottate per la comunicazione tra nodi della rete.
- Il sistema dovrà essere il più possibile robusto agli errori.
- La progettazione architetturale deve prevedere un meccanismo che permetta Avvio del sistema e Terminazione ordinata.
- L'architettura distribuita deve sottostare a vincoli temporali propri di un sistema ferroviario.
- La struttura distribuita deve essere tale da permettere estendibilità e scalabilità.

2.1.1 Gestione del Tempo

La simulazione è scandita da orari di partenza e di arrivo dei Treni che circolano tra le stazioni. Per questo è importante dotare il sistema di un *riferimento temporale* adeguato, che permetta di gestire i ritardi introdotti dalla comunicazione di rete, o dalla diversità di sincronizzazione degli orologi dei diversi nodi della rete.

Tale problema assume forme diverse in base al grado di distribuzione scelto per le componenti che generano gli eventi caratterizzanti la simulazione. In particolare, un livello di distribuzione alto, che prevede ad esempio la collocazione di una entità Stazione per nodo della rete, richiederà un meccanismo di regolazione del tempo più complesso e delicato rispetto ad un sistema con un livello di distribuzione più contenuto, che preveda ad esempio una distribuzione di singole Regioni di simulazione.

La scelta del riferimento temporale diviene quindi cruciale per lo svolgersi della simulazione. Abbiamo due tipi possibili di orologio:

- Assoluto: Prevede l'esistenza di un'entità dalla quale le varie componenti attingono per ottenere l'informazione temporale.
- Relativo: Ciascun nodo di calcolo possiede un proprio riferimento temporale interno.

É chiaro che la prima soluzione non si presta ad essere utilizzata per il problema presentato. Infatti esso, possedendo un flusso coninuo interno del tempo, non permetterebbe a entità indipendenti su nodi diversi di eseguire logicamente allo stasso istante (ad esempio due treni che in nodi diversi partono contemporaneamente da una stazione)

2.1.2 Acquisto di un Biglietto

In base al grado di distribuzione della modellazione realizzata, è necessario prevedere una struttura distribuita di biglietterie, in quanto la natura del problema prevede un livello di conoscenza globale soprattutto per l'acquisto di biglietti di treni a prenotazione.

2.1.3 Terminazione del Sistema

La durata di una simulazione di un sistema ferroviario è per sua natura indefinita. É quindi necessario un intervento esterno che ne decreti la terminazione. La *Terminazione del sistema* globale deve essere coordinata tra tutte le componenti distribuite, ed effettuata in modo tale da non far terminare l'esecuzione in uno stato inconsistente. Dovrà inoltre garantire che nessun nodo di calcolo rimarrà attivo (ad esempio, nessun thread in esecuzione o in attesa).

2.1.4 Avvio del Sistema

L'*Avvio del sistema* dove essere progettato in modo tale da permettere a tutte le componenti distribuite di interagire, ed evitare errori. In particolare:

- dev'essere previsto un meccanismo che permetta una rapida individuazione dei nodi con i quali ciascuna entità coopera;
- devono essere evitati (o gestiti) errori causati dal tentativo di comunicazione di thread concorrenti con entità non ancora pronte o allocate.

2.2 Concorrenza

Nell'analizzare le problematiche di concorrenza, è necessario individuare quali saranno le entità che svolgeranno un ruolo attivo all'interno della simulazione, e quali un ruolo reattivo, in base alle azioni compiute dalle entità attive. Nella simulazione di un sistema ferroviario, ho individuato le seguenti entità attive:

- Treno
- Viaggiatore

mentre le entità reattive principali saranno:

- Segmento
- Piattaforma

2.2.1 Ingresso in un Segmento da parte di un Treno

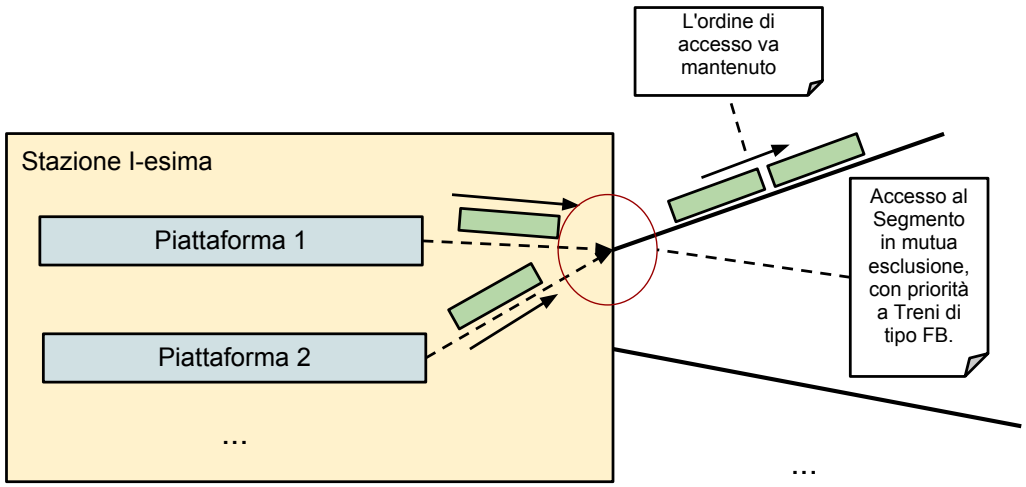

Figura 2.1: Accesso ad un Segmento da Parte di uno o più Treni.

L'accesso ad un segmento di collegamento tra due stazioni da parte di un Treno, è inerentemente concorrente. Tale azione presenta infatti i seguenti requisiti:

- L'ingresso presso un Segmento deve avvenire in *mutua esclusione*; è infatti impossibile che due o più entità Treno accedano ad uno stesso Segmento contemporaneamente.

- Più Treni possono circolare su un Segmento contemporaneamente. Questo comporta:
  - il mantenimento di un ordine di ingresso al Segmento;
  - la regolazione della velocità di transito di ciascun Treno in base alla velocità di quelli che lo precedono;
  - l'impossibilità di un Treno di accedere ad un Segmento qualora vi siano altri Treni che lo percorrono in senso opposto.
- Dev'essere data precedenza d'accesso al Segmento, ai Treni di tipo `FB`.

2.2.2 Uscita da un Segmento e accesso alla Stazione

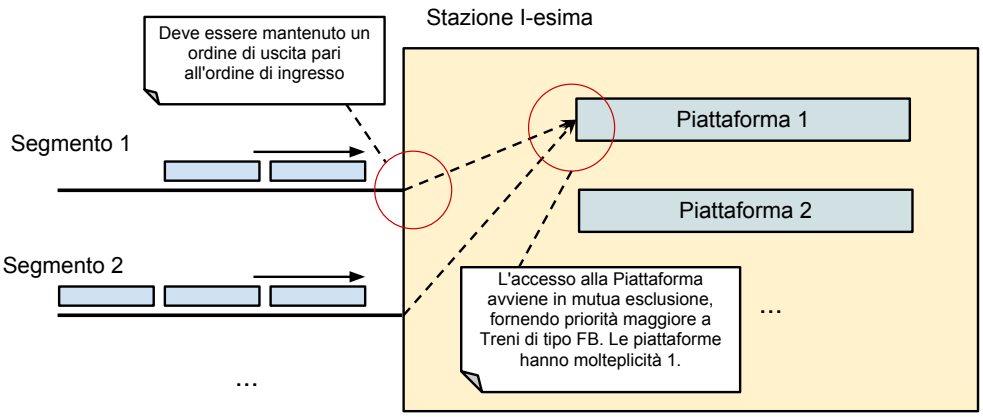

Figura 2.2: Uscita da un Segmento e accesso ad una Piattaforma di uno o più Treni.

Il problema introdotto in sezione 2.2.1, vincola i requisiti che dovranno essere soddisfatti relativamente all'uscita da un Segmento e conseguente accesso alla Stazione successiva. Per quanto riguarda l'uscita dal Segmento avremo quindi i seguenti requisiti:

- L'ordine di ingresso al Segmento dev'essere mantenuto all'uscita.
- L'ordine di uscita da un Segmento dev'essere mantenuto nell'accesso alla stazione successiva da parte dei Treni in transito.

L'ingresso in una Stazione, permette ad un Treno di occupare una della Piattaforme disponibili e, dato che da specifica una stazione può essere raggiunta da più Segmenti, tale azione sarà svolta in modo concorrente tra i treni in uscita dai vari Segmenti, secondo i seguenti vincoli:

- I Treni di tipo `FB` avranno priorità maggiore nell'occupare una Piattaforma.
- Ciascuna Piattaforma è acceduta in mutua esclusione, e può essere occupata da un solo Treno alla volta.

Questo significa che l'accesso ad una Piattaforma sarà ordinato, per tutti i Treni provenienti dallo stesso Segmento, in base all'ordine di uscita da quest'ultimo, e concorrente tra Treni provenienti da Segmenti diversi.

2.2.3 Acquisto di un Biglietto da parte di un Viaggiatore

Ciascun Viaggiatore deve acquistare un biglietto prima di poter usufruire dei servizi ferroviari. Per fare ciò, all'interno ogni stazione vi è una Biglietteria presso la quale l'acquisto può essere effettuato. Un biglietto sarà composto da una serie di Tappe, ciascuna relativa ad un tratto del percorso da portare a termine con uno specifico Treno, sia Regionale che FB. L'assegnazione di un Biglietto ad un Viaggiatore è semplice se il suo percorso prevede solo l'utilizzo di Treni Regionli, mentre è più complesso se vi sono tappe da raggiungere con Treni FB, i cui biglietti vengono erogari se e soltanto se vi è ancora posto all'interno del treno. É quindi necessario prevedere l'esistenza di una *biglietteria centrale* che mantenga le prenotazioni dei Treni di tipo FB. Si ricavano quindi i seguenti requisiti:

- La biglietteria centrale va acceduta in mutua esclusione, in modo da evitare prenotazioni inconsistenti di Biglietti.
- L'erogazione di biglietti può fallire in caso non vi siano posti per percorrere alcune tappe; il sistema deve reagire di conseguenza.

Capitolo 3

Soluzione

Di seguito, verrà presentata la soluzione realizzata, in termini di progettazione come sistema distribuito e concorrente.

3.1 Logica di Distribuzione

Un diagramma infromale delle componenti distribuite che compongono il sistema è presentato in figura 3.1. Di seguito verranno descritte le principali.

3.1.1 Regioni

La simulazione è stata suddivisa in *Regioni*, le quali risiederanno su nodi di calcolo diversi. Questa scelta aggiunge i seguenti requisiti minimi alla specifica iniziale:

- I Treni, se previsto dal percorso, possono viaggiare da una Regione all'altra.
- I Passeggeri possono raggiungere destinazioni in Regioni diverse.
- Esisteranno punti di collegamento che permettono a Treni e Passeggeri di raggiungere Regioni diverse.
- Deve essere garantita consistenza temporale nel passaggio da una Regione ad un'altra.

Da questa scelta consegue inoltre l'introduzione di un semplice *Server dei Nomi* che mantiene traccia di ciascuna Regione, in modo tale da rendere agevole la risoluzione della locazione alla quale l'entità si trova.

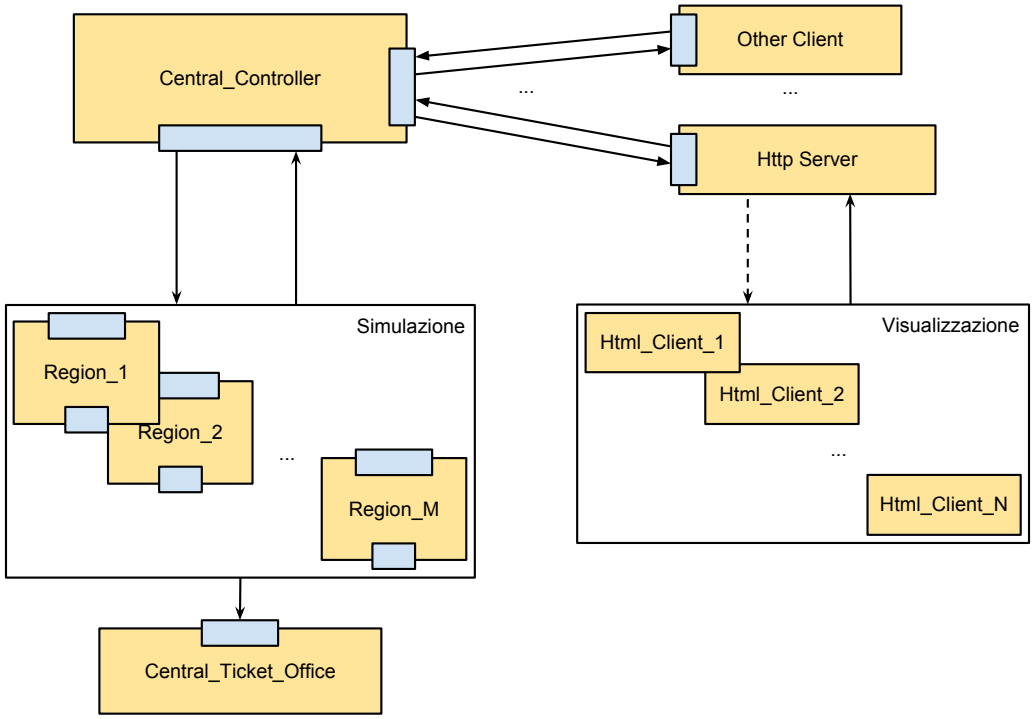

Figura 3.1: Diagramma informale che presenta una progettazione logica delle componenti distribuite.

Stazioni di Gateway

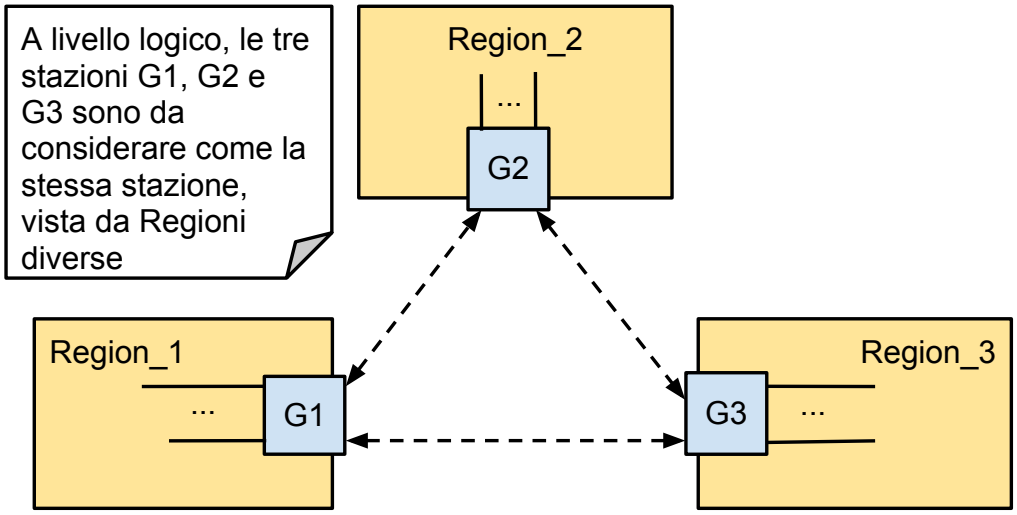

Figura 3.2: Utilizzo di stazioni di Gateway per collegare tre Regioni.

Nella progettazione di un simulatore distribuito su più Regioni, è necessario prevedere uno o più punti di collegamento tre di esse. A tal proposito, il progetto prevede *Stazioni di Gateway* (`Gateway_Station`), che permettono di raggiungere determinate Regioni remote. Le Stazioni di Gateway che collegano N Regioni in modo diretto, sono da considerare come un'unica Stazione logica, e ciascuna delle N Stazioni può essere vista come una porzione della Stazione globale accessibile dalla Regione corrente. Una rappresentazione dell'idea generale è visibile in figura 3.2.

Da tale assunzione, ricaviamo la seguente conseguenza:

Conseguenza 1 *Per ogni coppia G e G' di Stazioni di Gateway connesse (ovvero mutuamente raggiungibili in modo diretto) appartenenti a Regioni diverse rispettivamente R_G ed $R_{G'}$, $\nexists$ altra Stazione di Gateway $G'' \in R_{G'}$ tale per cui G'' è connesso a G.*

Per evitare sincronizzazioni tra thread distribuiti, sono posti dei vincoli nella struttura interna di questo tipo di stazoni. Ciascuna Stazione di Gateway G sarà dotata di un numero M di Piattaforme $\{P_1, ..., P_M\}$. Le Piattaforme saranno tali da permettere un *unico senso di percorrenza* delle Piattaforme, ovvero:

- Le prime $\{P_1, ..., P_K\}$ saranno disponibili all'accesso dalla Regione in cui G risiede.

- Le Piattaforme $\{P_K + 1, .., P_M\}$ saranno invece dedicate all'accoglimento di Treni provenienti da nodi diversi.

.

Questa scelta, vincola così il numero di Piattaforme per Stazione di Gateway:

Conseguenza 2 *Date N Stazioni di Gateway connesse, sia K_i il numero di Piattaforme che permettono ad un Treno di lasciare la Regione i-esima; allora ciascuna Stazione dovra complessivamente mantenere un numero di Piattaforme pari a* $\sum_{i=1}^{N} K_i$.

Tali vincoli sono sufficienti a mantenere locale ai nodi la sincronizzazione tra i thread che permettono l'esecuzione dei Treni. Una descrizione dettagliata del protocollo di accesso alle Stazioni di Gateway e di migrazione dei Treni, e presentata in sezione 3.3.2. Per una descrizione dell'entità e del ruolo delle Piattaforme, si rimanda alla sezione 3.2.4

3.1.2 Biglietterie

Per poter gestire meglio la definizione di un percorso e l'erogazione di un Biglietto per un Viaggiatore, ho pensato di introdurre una gerarchia su due livelli, di Biglietterie. Ci saranno dunque tre categorie di Biglietterie:

Biglietterie di Stazione
: Forniscono un'interfaccia adeguata ai Viaggiatori per poter acquistare un biglietto.

Biglietterie Regionali
: Hanno conoscenza regionale della topologia del grafo composto da Stazioni e Segmenti.

Biglietteria Cantrale
: Ha conoscenza di più alto livello; in particolare, essa mantiene traccia delle connessioni tra le varie regioni (ovvero i collegamenti tra Stazioni di Gateway di regioni diverse).

3.1.3 Controller Centrale

Il Conterollo Centrale è una entità distribuita, alla quale tutti i nodi inviano Eventi per notificare lo stato di avanzamento globale della simulazione. Esso fornisce una interfaccia alle varie Regioni per ricevere gli Eventi di simulazione, ed un'interfaccia per permettere a client remoti di poter visualizzare gli

effetti di tali Eventi. Quest'ultima possibilità è ottenuta mediante un meccanismo di tipo Publish/Subscribe, attraverso il quale client remoti possono registrarsi presso il Controller per ricevere, in modalità Push, gli Eventi. In questo modo è possibile per un qualsiasi client interfacciarsi al Controller e fornire, ad esempio, una rappresentazione grafica della simulazione.

Il Controller sarà anche responsabile dell'invio di un *segnale di terminazione*, che verrà recapitato a tutti i nodi che componegono la simulazione, mediante un algoritmo di Snapshot Distribuito, descritto in sezione 3.3.3.

3.2 Logica di Concorrenza

Internamente a ciascuna Regione di simulazione, sono definite entità che modellano le interazioni previste dalla specifica del problema. Di seguito sono descritte le principali.

3.2.1 Segmento

Un Segmento (`Segment`) è modellato come una *entità reattiva ad accesso mutuamente esclusivo, a molteplicità N*, dove `N` è il numero massimo di utilizzatori, la quale fornisce un'interfaccia utilizzabile da entità attive di tipo Treno per accedere ed uscire in maniera ordinata. Esso è caratterizzato da:

- le due stazioni che esso collega;
- una lunghezza;
- una velocità massima di percorrenza.
- un flag booleano `Free` che indica lo stato della risorsa, occupata o no.

3.2.2 Viaggiatore

In prima analisi, un Viaggiatore (`Traveler`) può essere modellato come una *entità Attiva* che esegue una sequenza di operazioni semplici. La modellazione dell'entità Viaggiatore come Attiva non può però essere semplicemente associata ad un processo, soprattutto in presenza di un modello di distribuzione come quello presentato in sezione 3.1.1. In questa ipotesi infatti, nel caso in cui un Viaggiatore si spostasse da una Regione ad un'altre, avremmo a disposizione solo due possibili soluzioni:

- La migrazione del processo che rappresenta il Viaggiatore sul nodo (Regioni) di destinazione, come distruzione del processo sul nodo di partenza e creazione dinamica dello stesso sul nodo destinazione. Questa operazione è in generale computazionalmente molto costosa.
- La replicazione del processo che rappresenta il Viaggiatore su tutti i nodi, e l'attivazione, intesa come cambio di stato del processo in modo tale che possa competere per la CPU; tale soluzione è molto costosa in termini di memoria utilizzata, e non scala all'aumentare del numero di passeggeri.

La soluzione adottata consiste nel disaccoppiare le operazioni svolte da ciascun Viaggiatore dal processo che le esegue, prevedendo una struttura dati (`Traveler_Pool`) costituita da:

- un *pool di M processi* dimensionato in maniera opportuna;
- una *coda di operazioni* che man mano vengono estratte ed eseguite dai processi nel pool.

In questo modo è sufficiente replicare per ciascun Viaggiatore, su tutti i nodi che compongono il sistema, una struttura dati che contiene i suoi dati e le operazioni che esso eseguirà, e cioè un Descrittore (`Traveler_Descriptor`). Il cambio di Regione di un Viaggiatore sarà quindi potrà essere ottenuto semplicemente inserendo la prossima operazione da eseguire per il Viaggiatore nella coda del pool di processi del nodo destinazione.

3.2.3 Treno

Un Treno (`Train`) è una *entità Attiva*, la quale esegue ciclicamente un numero finito di operazioni. Ciascuna entità Treno può appartenere a due categorie, FB a priorità più alta, e Regionale a priorità minore, ed è caratterizzata da:

- un identificativo univoco;
- una capienza massima;
- una velocità massima raggiungibile;
- una velocità corrente;

Anche in questo caso come per le entità di tipo Viaggiatore, ciascuna entità non viene mappata su un singolo processo. Infatti anche per le entità di tipo Treno ciò comporterebbe una complessa e costosa, in termini computazionali e di memoria, gestione del passaggio da un nodo (Regione) ad un altro. Ho optato invece nel progettare una struttura dati `Train_Pool` che mantiene:

- una coda di Descrittori di Treno, che contengono tutte le informazioni che distinguono una entità Treno;
- un pool di processi a dimensionato in maniera opportuna; ciascun processo sarà tale per cui una volta ottenuto un descrittore dalla coda, eseguirà per lui una fissata sequenza di azioni.

In questo modo un entità Treno eseguirà all'interno di un nodo se e soltanto se il suo descrittore sarà inserito nella coda di descrittori della struttura dati descritta.

3.2.4 Stazione

Una Stazione (`Station`) è modellata come una struttura dati contenete:

- un certo numero $n > 2$ di Piattaforme (`Platform`);
- una Biglietteria (`Ticket_Office`);
- un Pannello Informativo (`Notice_Panel`).

Essa offre una interfaccia alle entità Treno e Viaggiatore per l'accesso a Piattaforme e Biglietteria.

Piattaforma

Una Piattaforma è modellata come una *entità reattiva ad accesso mutuamente esclusivo, a molteplicità 1.* Essa espone un'interfaccia che permette alle entità Treno di potervi sostare ed effettuare discesa e salita delle entità Viaggiatore o di poter superare la stazione, e alle entità Viaggiatore di accodarsi in attesa di uno specifico Treno. Internamente essa mantiene due code di `Traveler_Descriptor`:

- `Leaving_Queue`, che conterrà i descrittori dei Viaggiatori in attesa di Treni per poter effettuare la partenza.
- `Arrival_Queue`, che conterrà i descrittori dei Viaggiatori in attesa di Treni per poter effettuare l'arrivo alla stazione.

Inoltre contiene un flag booleano `Free` che ne indica lo stato di occupazione. Ciascuna Piattaforma, in base al tipo di Stazione avrà percorrenza bidirezionale o uniderezionale.

Biglietteria

Una Piattaforma è modellata come una interfaccia che permette al Viaggiatore di acquisire un Biglietto di viaggio.

Pannello Informativo

Una Piattaforma è modellata come una *entità reattiva con agente di controllo, a molteplicità 1.* Esso espone una interfaccia tale da permettere alla stazione di notificare lo stato delle entità Treno che stanno arrivando, quelle in sosta e quelle in partenza.

3.3 Interazione tra le Entità

3.3.1 Percorrenza di un Viaggiatore

Alla luce della definizione dell'entità Viaggiatore in sezione 3.2.2, è possibile definire le azioni che compongono il viaggio:

- Acquisto di un Biglietto (`Ticket`) presso la Biglietteria della Stazione (`Ticket_Office`) di partenza. Ogni Biglietto è composto da una sequenza ordinata di Tappe (`Ticket_Stages`), ciascuna contenente:

  - `start_station`
  - `next_station`
  - `train_id`
  - `start_platform`
  - `destination_platform`

  e da un indice della prossima tappa del Percorso (`Next_Stage`).

- Una volta ottenuto un Ticket, vengono eseguite le seguenti operazioni per ciascuna Tappa del Biglietto:

  – Accodamento presso la Piattaforma `start_platform` della Stazione `start_station` in attesa del Treno `train_id`.

  – All'arrivo del treno `train_id`, Accodamento presso la Piattaforma `destination_platform` della Stazione `next_station` in attesa dell'arrivo di `train_id`.

Le azioni sopraelencate possono essere incapsulate in strutture dati che rappresentano una specifica operazione. In questo modo viene generata una *gerarchia di operazioni*, tutte derivanti da un'unica *operazione generica*. Di seguito identificheremo tali operazioni rispettivamente con `BUY_TICKET`, `LEAVE` e `ARRIVE`.

Il protocollo di operazioni che vengono eseguite da un Viaggiatore, è stato mantenuto il più semplice possibile, e l'intero percorso viene regolato dagli eventi generati dalle entità Treno alla partenza e all'arrivo dalle/nelle Stazioni.

Acquisto di un Biglietto

TODO ...

L'ultima azione eseguita dall'Operazione di acquisto di un biglietto consiste nel caricare nella coda della struttura dati `Traveler_Pool` descritta nella sezione 3.2.2, l'operazione `LEAVE` del Viaggiatore corrente.

Partenza da una Stazione

Denominiamo t la tappa di indice `Next_Stage`, ovvero la tappa corrente. Perché l'operazione di Partenza dalla stazione corrente venga effettuata, è necessaria la precondizione per la quale:

- l'operazione `LEAVE` è stata inserita nella apposita coda di `Traveler_Pool`;
- in qualche momento essa venga prelevata (rimossa) da tale luogo ed eseguita da uno dei thread nel Pool della struttura.

Le azioni compiute dall'operazione `LEAVE` sono:

- Estrazione `start_station` da t;
- Tramite l'interfaccia esposta dalla Stazione `start_station` (una procedura che identifichiamo con `Wait_For_Train_To_Go`), viene richiesto di attendere presso la Piattaforma `start_platform`.

Internamente alla Stazione, tale richiesta viene tradotta nell'inserimento del Viaggiatore, o meglio del suo Descrittore `Traveler_Descriptor`, nella coda `Leaving_Queue` della Piattaforma indicata da `start_platform`.

Arrivo alla Destinazione successiva

Similmente a quanto descritto per la partenza, l'arrivo a destinazione prevede che l'operazione `ARRIVE` sia già stata inserita all'interno della coda di operazioni in `Traveler_Pool`, e che un thread del pool la esegua (rimuovendola dalla coda).

Tale operazione prevede le seguenti azioni:

- Estrazione `next_station` dalla tappa corrente t.
- Tramite l'interfaccia esposta dalla Stazione `next_station` (procedura identificata con), viene effettuata una richiesta di attsa presso la Piattaforma `destination_platform`.

Internamente alla Stazione, la richiesta si traduce nell'inserimento del Descrittore del Viaggiatore corrente all'interno della coda `Arrival_Queue` interna alla Piattaforma `destination_platform` selezionata.

3.3.2 Percorrenza di un Treno

A ciascuna entità Treno, è assegnato un Percorso (`Route`) di andata e un Percorso di ritorno, ovvero sequenze di Tappe (`Stage`) successive, ciascuna composta dalla tupla:

```
<next_segment,next_station,next_platfom,actio,region>
```

dove `action` indica quello che un Treno dovrà compiere presso la prossima stazione, a scelta tra `ENTER`, per entrare ed effettuare discesa e salita passeggeri o `PASS` per non fermarsi e oltrepassare la Stazione; il campo `region` invece, indica la regione di destinazione, e viene utilizzato nel caso di transito su stazione di Gateway.

Le operazioni effettuate sono, per ciascuna Tappa del Percorso corrente (di andata o di ritorno):

- Accesso al prossimo Segmento `next_segment` previsto.
- Percorrenza all'interno del Segmento come attesa finita di durata proporzionale alla lunghezza del Segmento e alla velocità massima alla quale il Treno può percorrerlo.
- Uscita dal Segmento e richiesta di Accesso alla Stazione successiva (`next_station`) presso la Piattaforma indicata da `next_platfom`, per eseguire l'azione `action`.
- Se `action = ENTER` allore effettua discesa e salita dei Viaggiatori in attesa dell'arrivo del Treno.
- Uscita dalla Piattaforma `next_platfom`.
- Se ci sono ancora Tappe da percorrere nel percorso (`Route`), allora il Descrittore del Treno viene inserito nella coda di `Train_Pool`.

Accesso ad un Segmento

L'accesso alla risorsa protetta Segmento è regolato da una interfaccia ben definita, che permette:

- Ingresso.
- Uscita.

Per mantenere l'ordine di accesso, ciascun Segmento è dotato di una Coda FIFO (`Train_Queue`) che conterrà i Descrittori dei Treni correntemente in transito.

Ingresso

La richiesta di accesso (`Enter`) al segmento avviene in mutua esclusione tra tutti le entità Treno. In ogni momento quindi solo una entità eseguirà all'interno della risorsa protetta. Una volta ottenuta la risorsa ciascun Treno compie le seguenti operazioni

- Inserimento del Descrittore nella coda `Train_Queue`;
- Aggiornamento della velocità di percorrenza del Treno entrato in base a quella dei treni che lo precedono.
- Nel caso in cui la risorsa risulti vuota (viene consultato il flag booleano che mantiene questa informazione), allora viene modificato il valore del flag in modo tale da indicare lo stato occupato della risorsa, e memorizzata la stazione di provenienza del Treno.

Una volta terminate queste due operazioni, il Treno rilascia la risorsa e, basandosi sulla lunghezza e sulla velocità da mantenere, simula la percorrenza rendendosi inattivo (non competitivo per l'ottenimento della CPU) per un tempo dato dalla semplice equazione: $Time = Segment\_Length/Actual\_Speed$.

Nel caso in cui una volta ottenuta la risorsa protetta Segmento, un Treno richiedesse l'accesso nella direzione opposta a quella dei Treni che percorrono il Segmento (ovvero abbiamo la situazione in cui il flag booleano indica lo stato occupato, e la stazione di provenienza memorizzata presso il Segmento è diversa da quella del Treno richiedente) allora questo dovrà attendere fino a che il Segmento non sarà si sarà liberato dai treni in transito.

Una prima soluzione possibile è quella di prevedere una coda di attesa interna alla risorsa Segmento (`Waiting_Queue`), per i Treni provenienti dalla direzione opposta rispetto al senso di marcia corrente: tali Treni dovranno avere priorità maggiore nell'accedere al Segmento appena esso diventa vuoto, rispetto ad altri Treni che sopraggiungono successivamente.

Questa prima modellazione, non risulta però sufficiente a garantire che i Treni nella coda avranno accesso al Segmento in qualche momento: infatti vi è il rischio di portare a Starvation dei Treni in attesa. Si consideri la presenza di un percorso circolare composto da N Segmenti, e di M Treni che viaggiano lungo tale percorso in senso orario, in modo tale che in ogni istante ci sia almeno un treno all'interno di ciascun

Segmento. Se ora aggiungiamo al sistema un Treno che viaggia in direzione opposta, allora esso, adottando la semantica di accesso descritta non riuscirà mai a percorrere uno dei Segmenti.

Una possibile soluzione al problema, è mantenere nella coda `Waiting_Queue` tutte le entità Treno che non possono accedere correntemente, sia perché provenienti dalla direzione opposta, sia perché il numero massimo di accessi da una direzione è stato raggiunto. In questo modo la semantica di accesso viene modificata come segue:

- Se `Free = True` allora il Treno corrente
  - Modifica il valore di `Free` a `False`.
  - Imposta la Stazione provenienza con la propria.
  - Incrementa di 1 il contatore di accessi.
- Se invece `Free = False` allora
  - Se la stazione di provenienza del Treno corrente è diversa da quella memorizzata nel Segmento, allora inserisci il Treno nella coda `Waiting_Queue` e STOP.
  - Altrimenti
    - Se il contatore di accessi è $<$ del limite massimo incrementa di 1 il contatore di accessi.
    - Altrimenti accoda il Treno su `Waiting_Queue` e STOP.
- Se il Treno ha avuto accesso al Segmento, inserisci il Descrittore del Treno nella coda `Train_Queue` e modifica velocità di transito del Treno a seconda della massima velocità possibile.

Uscita

L'uscita (`Leave`) da una Segmento da parte di una entità Treno, ha come prerequisito l'aver avuto accesso al Segmento. É quindi possibile assumere che il descrittore del Treno che intende uscire dal Segmento sia presente all'interno della coda `Train_Queue`, e che inoltre al momento di tale richiesta abbia già terminato il tempo previsto di attesa che simula la percorrenza.

Il requisito principale richiesto dall'azione di uscita è che essa avvenga in maniera ordinata, in base all'ordine con cui i Treni hanno avuto accesso al Segmento. La soluzione apportata mira a garantire tale ordine di uscita, senza fare alcuna assunzione sull'ordine con il quale i Treni verranno scelti per l'esecuzione dallo scheduler. La semantica adottata è quindi la seguente:

- Viene controllato per prima cosa se il Treno corrente è effettivamente il prossimo che deve uscire secondo l'ordine di ingresso.
- Se è è il prossimo (vengono confrontati i Descrittori) allora il Treno:
  - Nel caso in cui la coda sia vuota,
  - può abbandonare la risorsa protetta.
- Altrimenti il Descrittore del Treno corrente viene posto in attesa su una coda; tale coda verrà riesaminata ogni volta che un Treno abbandonerà la risorsa protetta (disponendo ad esempio di instruzioni wait e signal, ciascun Treno uscente invocherà una signal sulla coda), e se il Treno in esecuzione non sarà nuovamente il Treno destinato ad uscire, esso verrà riaccodato.

La semantica descritta, garantisce ingresso sequenziale a molteplicità limitata, e uscita ordinata secondo l'ordine di ingresso. Il passo successivo consiste nel garantire l'accesso alla Stazione con lo stesso ordine di uscita per tutti i Treni provenienti dallo stesso Segmento.

Accesso ad una Stazione Regionale

I prerequisiti alla richiesta da parte di un Treno T di accesso ad una Stazione sono:

- Il Treno T ha avuto accesso ad un Segmento S che collega la stazione corrente a quella dalla quale proviene. Ciò significa che il suo Descrittore sarà stato inserito nella coda `Train_Queue` di S.
- Il Treno T ha simulato la percorrenza su S
- Il Treno T uscito dal Segmento S, quindi è stato rimosso dalla coda `Train_Queue` di S.

Ingresso Ordinato

Per tutti i Treni provenienti dallo stesso Segmento, è necessario mantenere un ordine di richiesta di accesso alla Stazione successiva uguale a quello utilizzato per l'uscita dal Segmento. Poiché il sistema non può fare assunzioni sulle scelte dello scheduler che verrà adottato, è necessario introdurre un meccanismo che riesca a mantenere l'ordine di richiesta di accesso anche nel caso in cui, il thread che rappresenta il

Treno appena uscito dal Segmento venga prerilasciato e venga eseguito il thread rappresentante il Treno successivo.

Ho vagliato diverse possibili soluzioni:

1. Rappresentazione della Stazione come entità passiva ad accesso mutuamente esclusivo. Essa manterrà al suo interno una coda per Segmento, del tutto simile alla coda `Train_Queue` interna ai Segmenti, e popolata parallelamente a `Train_Queue`. In questo modo sarà semplice garantire (con un meccanismo simile a quello utilizzato per l'uscita da un Segmento) un accesso ordinato. Questa soluzione ha però uno svantaggio evidente: la richiesta di accesso avverrà in mutua esclusione anche tra i Treni provenienti da Segmenti diversi, e che quindi si ritroveranno a dover superare un punto di sincronizzazione inutile, se diretti a Piattaforme diverse della Stazione.
2. Una seconda possibilità prevede la presenza, all'interno di ciascuna Stazione, di una risorsa ad accesso mututamente esclusivo per ciascun Segmento entrante, la quale si occuperà di garantire l'ordine di accesso (tale ordine sarà dato da una coda interna aggiornata in modo analogo alla soluzione precedentemente illustrata). Una volta che il Treno ha guadagnato il permesso di accedere in mutua esclusione alla risorsa protetta di controllo, esso verrà inserito su una coda di Treni presso la Piattaforma corretta. In questo modo la Stazione diventerà solo un'interfaccia utilizzabile per l'accesso alle piattaforme, e inoltre i vincoli di ordinamento e accesso concorrente alle Piattaforme da Treni provenienti da Segmenti diversi saranno garantiti.

   Lo svantaggio di utilizzare questa soluzione risiede nella creazione delle entità protette, una per ciascun Segmento entrante, che regolano l'accesso ordinato, e quindi spostano parte della conoscenza della topologia della rete ferroviaria anche sulle Stazioni, rendendone più complessa la configurazione iniziale.

Per la soluzione realizzata ho scelto la seconda opzione.

Ingresso in una Piattaforma

Dopo aver superato la risorsa di controllo, ciascun Treno viene accodato presso la Piattaforma, interna a ciascuna Stazione, prevista dal percorso. La semantica di accodamento su di una generica Piattafor-

ma P (supponiamo disponibile mediante procedura eseguita in mutua esclusione) è la seguente:

- Se P è libera, ovvero il flag booleano `Free` sarà `True`, allora
  - Il Treno occupa la Piattaforma (`Free = False`).
  - Se l'azione (`action`) prevista dal Percorso è `ENTER` allora:
    * esegue l'operazione di **Discesa dei Viaggiatori**;
    * rilascia la risorsa, in modo tale che altri Treni vi si possano accodare, e rimane in attesa fino all'orario prestabilito di partenza;
- Se invece la Piattaforma risulta occupata (`Free = False`), il Treno viene accodato presso una coda di Treni interna alla Piattaforma. Rappresentando la Piattaforma come una risorsa protetta da monitor, l'accodamento si traduce in una attesa su condizione, `wait(Free = True)`.

L'Operazione di **Discesa dei Viaggiatori**, si basa sulla possibilità che presso la Piattaforma corrente P vi siano Viaggiatori all'interno della coda `Arrivals_Queue` in attesa di un evento generato da uno specifico Treno; tale evento coincide con il suo arrivo presso P. L'operazione di Discesa dei passeggeri ha come precondizione l'accesso alla Piattaforma da parte di un Treno T descritto in precedenza, ed è realizzata dalle seguenti azioni (eseguite dal thread associato a T in mutua esclusione all'interno della risorsa protetta P):

- Viene estratto ciascun Viaggiatore V dalla coda `Arrival_Queue`, e per ciascuno di essi:
  - Se il Treno atteso da V (informazione recuperabile dalla Tappa corrente del suo Biglietto) è proprio T allora
    * il numero di posti occupati di T viene decrementato;
    * se la Stazione corrente S **non** è la destinazione che V deve raggiungere allora:
      · l'Operazione `LEAVE` del Viaggiatore viene inserita nella coda di operazioni di `Traveler_Pool`;
      · viene incrementato l'indice `Next_Stage` nel Biglietto del viaggiatore V.
  - Se T invece non è il Treno atteso da V, quest'ultimo viene reinserito nella coda `Arrivals_Queue`.

Uscita da una Piattaforma

L'uscita da una Piattaforma P necessita di due prerequisiti principali:

- Il thread che gestisce il Treno ha prima effettuato l'ingresso, e quindi `Free = False` ed è già stata effettuata la Discesa dei Passeggeri.
- É stato generato l'evento che notifica l'arrivo dell'orario previsto per la partenza.

A questo punto, le operazioni effettuate all'interno della sezione critica (eseguita in mutua esclusione dal thread corrente) sono:

- Il flag booleano `Free` viene ripristinato a `True`.
- Viene effettuata la Salita dei Viaggiatori in attesa presso P.
- La risorsa viene rilasciata dal Treno in esecuzione.

La **Salita dei Viaggiatori** è simile alla discesa dei Passeggeri presentata al punto precedente. Essa viene eseguita all'interno della risorsa protetta P (quindi in mutua esclusione), ed è composta dalle seguenti azioni:

- Viene estratto ciascun Viaggiatore V dalla coda `Leaving_Queue` di P, e per ciascuno di essi:
  - Se il Treno atteso da V (informazione recuperabile dalla Tappa corrente del suo Biglietto) è proprio il Treno correntemente in esecuzione T e *se la capienza massima di T non è stata raggiunta* allora:
    - il numero di posti occupati di T viene incrementato;
    - l'Operazione `ENTER` del Viaggiatore viene inserita nella coda di operazioni di `Traveler_Pool`;
  - Se T invece non è il Treno atteso da V, quest'ultimo viene reinserito nella coda `Leaving_Queue`.

Accesso ad una Stazione di Gateway

Alcuni Treni seguono percorsi che attraversano più Regioni. Tra le tappe che compongono tali percorsi, alcune indicheranno l'attraversamento di Regioni di Gateway, introdotte nella sezione 3.1.1. L'azione di ingresso a tali stazioni segue le stesse regole descritte per le stazioni Regionali nella sezione 3.3.2,

e viene quindi effettuata localmente alla singola Regione (nodo) di provenienza. Le azioni di Discesa e Salita dei Viaggiatori, se indicate dal valore `ENTER` del parametro `action` della prossima Tappa, vengono invece ripetute su entrambe le Regioni sulle Piattaforme corrispondenti. Questo comporta che sarà effettuata:

- La discesa dei Viaggiatori in attesa di arrivare alla stazione S presso la Regione R sulla Piattaforma P, con il Treno corrente T.
- La salita dei Viaggiatori che attendono presso la Piattaforma P il Treno T, in una Regione o nell'altra.

Il Passaggio di un Treno da una regione all'altra può essere schematizzato come segue: sia $G1$ la Stazione di Gateway che connette la regione $R1$ ad $R2$, con $G1$ appartente ad $R1$. Un Percorso che attraversa i due Gateway conterrà almeno una Tappe T tale per cui il campo `next_station` avrà il valore $G1$, e come `destination_platform` una delle Piattaforme possibili per effettuare l'accesso, mentre il campo `region` conterrà invece la *Regione di destinazione*. Le operazioni eseguite saranno le seguenti:

- Il Treno corrente T effettua l'accesso alla Stazione $G1$, Piattaforma P.
- T, se previsto dal Percorso, effettua Discesa dei Viaggiatori.
- Il thread che gestisce T viene interrotto, ovvero viene modificato il flusso di esecuzione delle operazioni cicliche da esso eseguite.
- Il descrittore del Treno corrente D_T viene serializzato (*marshalling*), e inviato tramite invocazione remota alla Stazione di Gateway della Regione $G2$ ad essa connessa. L'individuazione dell'indirizzo della regione specificata nel parametro `region`, avviene interrogando il `Name_Server`, se esso non è già presente in una cache locale.
- Presso la regione $R2$, stazione di Gateway $G2$, vengono eseguite le seguenti operazioni:
  - Il Descrittore viene de-serializzato (*unmarshalling*).
  - I dati del Descrittore $D_{T'}$ presenti nella regione $R2$ vengono aggiornati con quelli ricevuti.
  - Viene ottenuta la Piattaforma in mutua esclusione, in modo da effettuare, se previsto dal Percorso, la Discesa dei Viaggiatori, e ne viene impostato il campo `Free` a `False`.

– Vengono operate attesa fino all'orario di partenza previsto (secondo il clock del nodo corrente), Salita dei Viaggiatori e Partenza dalla Stazione come per le Stazioni Regionali.

Si noti che data la conseguenza 1, questa soluzione garantisce una risoluzione univoca della Stazione di Gateway di arrivo.

3.3.3 Terminazione distribuita

Capitolo 4

Tecnologie Adottate

La scelta delle tecnologie per l'implementazione del prototipo realizzato, è stata operata nell'ottica di individuare strumenti che permettessero una agevole attuazione della soluzione presentata.

Sono stati utilizzati principalmente due linguaggi di programmazione.

Scala
Il linguaggio Scala è un linguaggio funzionale Object Oriented. Ho utilizzato questo linguaggio per la realizzazione di

- **Controller Centrale**
- **Biglietteria Centrale**
- **Application Server**, al quale i client HTTP possono collegarsi mediante protocollo Web Socket per poter fornire una rappresentazione grafica della simulazione, in linguaggio HTML e Javascript. Per la sua realizzazione è stato utilizzato il framework MVC `Play 2.0` (disponibile all'indirizzo `http://www.playframework.com/`).

Le interazione tra thread interne alle componenti distribuite, sono state gestite adottando il modello di concorrenza ad Attori nativo.

Ada
Il linguaggio Ada è stato utilizzato per realizzare il core di simulazione presente in ciascuna regione. Il modello di concorrenza fornito dal linguaggio è risultato più adatto per rappresentare le interazioni tra le entità descritte al capitolo precedente rispetto ai modelli offerti da altri linguaggi. Il modello di concorrenza ad Attori di Scala, per esempio, avrebbe reso infatti necessaria l'adozione di entità attive Server a

protezione delle entità reattive come Segmenti e Piattaforme. Questo comporta:

- Maggiore utilizzo di thread, per poter eseguire gli attori a protezione delle entità;
- Maggiore complessità di terminazione;

Tuttavia avrebbe avuto il vantaggio di fornire nativamente il meccanismo descritto in sezione **??**.

Ada ha fornito un alto livello di controllo di thread e risorse protette, e il suo sistema di tipi molto restrittivo ha consentito di ridurre i possibili errori in fase di sviluppo.

L'interazione tra componenti remote realizzate con tecnologie omogenee, è stata possibile mediante il middleware yami4 (`http://www.inspirel.com/yami4/`), compatibile, tra l'altro, con Ada e Java (e quindi di conseguenza anche con Scala).