Analisi critica sul progetto di Sistemi Concorrenti e Distribuiti

28 Ottobre 2010

Sommario

Analisi del progetto didattico di Sistemi Concorrenti e Distribuiti. Considerazioni sulle scelte adottate e confronto con la soluzione proposta in corso di colloquio.

Informazioni documento

Nome file	AnalisiCritica.pdf
Versione	1.0
Distribuzione	Prof. Vardanega Tullio
	Miotto Nicola
	Nesello Lorenzo

Indice

1 Introduzione

Questo documento ha come scopo quello di fornire una analisi critica del progetto svolto per il corso di Sistemi Concorrenti e Distribuiti che prevedeva lo studio e la prototipazione di un simulatore di formula 1. Tale documento è strutturato definendo il problema e la soluzione suggerita nel corso del colloquio orale per procedere poi con una analisi dei passi fatti per giungere alla nostra soluzione del problema discutendo e motivando le decisioni prese. Nel corso del documento verranno messi in evidenza i punti di divergenza fra le due soluzioni per concludersi con una discussione critica delle nostre scelte.

2 Il problema / La soluzione attesa

Senza scendere nei dettagli del progetto didattico (già visti nei documenti presentati precedentemente) il problema si può ridurre ad una mappatura di una competizione di formula uno in un sistema composto da entità concorrenti e risorse condivise, nonchè nodi distribuiti. Si presentava necessario, quindi, identificare ogni entità protagonista di una gara di formula 1 come entità presente in un sistema concorrente e distribuito e sfruttare le sue caratteristiche intrinseche a vantaggio della simulazione.
Dalla discussione fatta in corso di colloquio abbiamo appreso che sarebbe stato auspicabile che un sistema come quello accennato fosse ottenuto sfruttando quanto più possibile le caratteristiche delle primitive offerte da un linguaggio per simulare la realtà. Vale a dire, associando piloti a thread concorrenti, il circuito ad una serie di risorse condivise, la possibilità di sospendere processi per simulare l'attraversamento di un tratto, i box come nodi distribuiti e i problemi di rete come problemi di comunicazione pilota/box. In pratica appoggiarsi a strutture già esistenti (che possano essere magari trovate in un linguaggio specifico), implementando solamente le feature aggiuntive per avere la simulazione richiesta. A soluzione data, si è visto come progettare un tale sistema in ADA sarebbe risultato efficiente, efficace e più semplice (nonchè verificabile e comprensibile). Quanto prodotto nel corso dello sviluppo del nostro progetto, per quanto produca risultati corretti e prevedibili, non si può dire essere essenziale e semplice. Il risultato è stato frutto di una serie di ragionamenti mirati a obbiettivi forse diversi da quelli attesi e, dal lato negativo, da un modello di ragionamento diverso da quello previsto, volto a trattare il problema dato in modo più generico. Vediamo ora quali avrebbero potuto essere i punti in fase di analisi e progettazione che potrebbero aver portato alla divergenza delle due soluzioni.

3 La nostra analisi del problema

3.1 Definizione obbiettivo del simulatore

Durante la fase iniziale del progetto, si è pensato a cosa esattamente si volesse ottenere dal prototipo del simulatore. Si è posto come requisito primario del prodotto da sviluppare la massima predicibilità, per quanto possibile. Si è pianificato quindi di utilizzare la prima fase dello studio della soluzione per progettare un simulatore che potesse produrre risultati deterministici. Quello che ci si aspettava era, in pratica, una funzione. Si è deciso di dedicare ad una seconda fase l'introduzione di elementi che conferissero un certo livello di non determinismo alla simulazione. Questo per due motivi:

- una volta sviluppato il sistema in modo tale che fosse totalmente predicibile (o in cui comunque gli elementi di non predicibilità fossero controllabili senza assunzioni troppo forti), sarebbe risultato meno complesso verificarne la correttezza. Dopo questo passo sarebbe stato possibile introdurre delle componenti per dare più realismo (fattori che influenzino la gara in modo stocastico)
- in secondo luogo si è pensato che l'utilizzatore del simulatore avrebbe potuto desiderare un sistema che, qualora richiesto e per quanto possibile, producesse output predicibile (esempio: data una configurazione di auto e una pista, l'ordine di arrivo e i tempi devono essere uguali per qualunque esecuzione)

3.2 Studio delle entità di sistema

La strada più intuitiva per creare una relazione fra sistema simulato e realtà è quella di assumere che la concorrenza dei thread possa rappresentare le vicende dei piloti durante la gara.
Inoltre, fissando un circuito come entità passiva ad accesso multiplo si sarebbe riprodotto il comportamento reale dei piloti nel momento in cui provano ad attraversare un tratto. Seguendo lo stesso ragionamento si è ipotizzato di poter, in un secondo momento, distribuire i box mappando la comunicazione radio fra box e concorrenti con la comunicazione di rete. Infine si è tenuto in considerazione di poter progettare delle TV remote da cui fosse possibile assistere alla gara.
Si è pensato al concetto di task/thread, o meglio, entità attiva, nella maniera più language-independent possibile, senza pensare alle primitive presenti in un preciso linguaggio di programmazione. Così facendo non si sono fatte assunzioni sull'ambiente di esecuzione. Questo ci ha permesso di arrivare a pro-

gettare un sistema implementabile in ogni linguaggio che offra le funzionalità di multithreading.

3.3 Analisi dei problemi legati a concorrenza e distribuzione

3.3.1 Il determinismo prima di tutto

Inizialmente si è studiata una soluzione che utilizzasse sospesioni relative per simulare l'attraversamento di tratti della pista. Successivamente si è valutato come l'idea avrebbe trovato una sua implementazione in un linguaggio come Ada. Tale soluzione però si è rivelata non praticabile in quanto la sospensione relativa nel nostro caso sarebbe stata influenzata dai prerilasci (ipotizzando che ogni auto eseguisse un loop per l'attraversamento sequenziale dei tratti della pista). Ci sarebbe stato un accumulo di ritardi (drift) col passare delle iterazioni. Dopo questo approccio si è passati a pensare ad una soluzione che prevedesse l'utilizzo di sospensioni assolute. Parallelamente si è cominciato a pensare all'influenza che una eventuale distribuzione avrebbe potuto avere sul sistema iniziando a modellare i primi scenari worst-case. Si è quindi iniziato a ragionare in un'ottica worst-case-driven. Volendo seguire la linea del massimo determinismo possibile, i ritardi non controllabili causati dalla comunicazione remota fra componenti non avrebbero favorito il raggiungimento di tale scopo se sul nodo centrale la simulazione fosse stata governata da un orologio assoluto. Non conoscendo a fondo i meccanismi offerti da Ada abbiamo anche ipotizzato a come si potesse risolvere il problema in altri linguaggi. Il nostro pensiero però è fortemente orientato alla filosofia Java dove il modello di concorrenza impedisce un controllo completo dello scheduling dei thread. Forti di questo ragionamento abbiamo approcciato il problema (sempre senza perdere di vista il determinismo da noi richiesto) ipottizzando un worst case molto simile a quello che si sarebbe potuto verificare nell'ambiente di esecuzione di altri linguaggi (Java su tutti) molto meno dotati dal punto di vista delle funzionalità offerte. Non vi è stata alcuna assunzione sulla potenza del calcolatore o sul numero di istruzioni macchina legate ad uno statement nel codice. E' sembrato che assunzioni del genere non fossero adatte alla modellazione di scenari generici.

3.3.2 Distribuzione e worst-case

In una soluzione come quella suggerita durante il colloquio le problematiche di distribuzione vanno trattate insieme a quelle di concorrenza. Nel nostro caso, invece, abbiamo cercato prima di formulare una soluzione che producesse una simulazione temporalmente coerente e completamente prevedibile in modo indipendente anche dalle scelte di distribuzione. La distribuzione è diventata

quindi un motivo di discussione per completare gli scenari worst case in fase di analisi e una problematica da discutere in dettaglio in una seconda fase della progettazione. Come già accennato, i ritardi di rete non sono una caratteristica su cui è possibile avere gran controllo.

3.4 Studio della soluzione

La soluzione è stata, come già detto, pensata per poter essere implementata in modo del tutto platform e language indepentent. L'utilizzo di flussi di tempo relativi per ogni entità concorrente ha permesso di poter astrarre completamente dall'ambiente sul quale il sistema sarebbe stato eseguito nonchè dalla presenza o meno di determinati strumenti nel linguaggio utilizzato. La differenza fra progettare una soluzione basata su un orologio di sistema e una basata su un'insieme di orologi relativi come la nostra è molto semplice: la prima sarebbe stata implementabile con sforzo relativamente basso con linguaggi adatti (adesso penseremo ad Ada). Implementarla con un linguaggio tipo Java avrebbe richiesto uno sforzo ben più alto. Non tanto per avere una specie di timer che permettesse di ragionare sui tempi di sistema, quanto più per riuscire a dare un controllo ai thread concorrenti. Le specifiche relative alla gestione del multitasking nella JVM di Java non sono date poichè, volendo essere un linguaggio portabile, esse sono strettamente correlate al sistema operativo sottostante. Per scelta progettuale quindi java non offre nessuna garanzia su ordinamento e scheduling dei processi. La seconda ha richiesto di progettare delle strutture dati (quali code ordinate sui tempi di arrivo) per permettere di applicare un controllo sui thread qualora non fosse stato possibile a livello di linguaggio. La soluzione ha chiaramente richiesto uno sforzo maggiore di quella suggerita in corso di colloquio, ma di sicuro sarebbe stata utilizzabile a prescindere dalla tecnologia e dal sistema dato. Da quanto abbiamo evinto dal colloquio, inoltre, la soluzione consigliata si basava anche su assunzioni dipendenti dall'architettura del sistema, dal numero di istruzioni macchina corrispondendi in media ad ogni statement di codice eccetera. La soluzione che abbiamo adottato non voleva essere dipendente da tale assunzione. Si è pensato che la complessità del sistema avrebbe potuto crescere indiscriminatamente (per esempio implementanto un'intelligenza artificiale a livello di pilota più elaborata) senza dover intaccare la correttezza temporale del simulatore.

3.5 Progettazione dell'architettura

Dopo aver studiato la soluzione adatta a venire incontro alle esigenze esplicate in precedenza, si è passati alla progettazione dell'architettura del sistema. Si è cercato nella prima fase di concepire il sistema e le sue componenti non con-

siderando l'esistenza di entità distribuite. La soluzione presentata permetteva infatti di non preoccuparsi delle problematiche di rete per garantire la correttezza temporale del simulatore.
In un secondo momento, con l'architettura del sistema sotto mano, si è deciso quali componenti sarebbe stato desiderabile che fossero distribuite. A questo punto, attuare la separazione remota delle entità sarebbe stato compito di facile attuazione. Si decise di prendere in considerazione l'ipotesi di un middleware che fornisse un protocollo di comunicazione adatto a governare la maggior parte dei problemi dovuti alla presenza della rete in modo da garantire il corretto funzionamento della competizione anche con la distribuzione in mezzo. Le scelte furono finalizzate a fornire maggior realismo alla soluzione (box distribuiti per problemi radio, tv perchè naturalmente dislocate in giro) ed eventualmente fornire maggiore efficienza (IA dislocata in nodi separati per scalare con l'aumento di complessità della stessa) .Come ultima motivazione distribuire il caricodi lavoro in più nodi in modo da alleggerire il server ospitante la competizione.

4 Considerazioni varie sulla nostra soluzione

4.1 Determinismo vs Non determinismo

La soluzione proposta durante il colloquio avrebbe garantito coerenza temporale per il sistema concorrente locale (su assunzione di istruzioni macchina eseguite, da noi non fatte) ma avrebbe inevitabilmente prodotto un non determinismo dato successivamente dai ritardi di rete che non prendono in considerazione l'orologio di sistema della competizione. Ai ritardi di rete si sarebbero sommati potenziali ritardi in base alla complessità dei calcoli effettuati dai box. La soluzione suggerita in fase di colloquio prevede che ogni task (concorrente) che gareggia, abbia la corsa scandita dall'orologio assoluto. Ogni istante di risveglio corrisponde all'istante reale in cui tale concorrente avrebbe intrapreso l'attraversamento del nuovo segmento. Per quanto riguarda i box (distribuiti), la nostra soluzione prevede che il concorrente instauri una comunicazione sincrona via radio col box per ottenere informazioni sul come procedere con la corsa. Comunicazione che fallirebbe solo in caso di mancanza di rete. E' lecito pensare che con la soluzione proposta durante il colloquio, invece, ciò non sia possibile: i ritardi dati da una RPC sono nell'ordine dei ms e potrebbero aumentare con condizioni di rete pessime. Il task che stia quindi calcolando l'istante di risveglio per l'esecuzione successiva verrebbe influenzato da questo ritardo. Nella realtà sarebbe come se il pilota si fermasse mentre comunica con il box per poi ripartire, il che non è plausibile. Sarebbe stato possibile, comunque, cambiare la direzione di comunicazione: il box comunica in modo asincrono eventuali modi-

fiche di strategia al concorrente. Il thread concorrente, in prossimità dei box o in qualunque altro momento, dovrebbe alterare la sua strategia in caso ci sia stata una comunicazione da parte dei box, e questo occuperebbe anche un numero limitato di istruzioni. Verrebbe tuttavia introdotto un livello di non determinismo che non si voleva avere inizialmente. Un ritardo potrebbe causare un mancato arrivo della nuova strategia al momento richiesto. Questo cambierebbe un esecuzione da quella precedente. Abbiamo pertanto scelto di progettare il sistema in modo che i casi come quello sopra citato venissero ridotti, fino a presentarsi solo nell'eventualità in cui la comunicazione di rete venisse tagliata; in questo caso si è scelto di far procedere la competizione in una direzione diversa, ma si sarebbe potuto comunque far attendere il concorrente fino alla riconnessione per garantire che la competizione producesse un risultato uguale a quelle simulate sotto le stesse condizioni.

4.2 Esperienza utente

Per quanto rigurarda l'esperienza utente, non è da tralasciare il fatto che si è puntato sulla correttezza delle informazioni presentate, fornendo la possibilità di velocizzare/rallentare a piacimento l'esecuzione. Il tempo che si vede scorrere negli schermi è comunque solidale con quello relativo alla competizione anche se non perfettamente mappabile con quello reale che scorre. Posto che il server centrale stia eseguendo la competizione secondo tempi assoluti e mappabili sullo scorrere del tempo reale il problema si sposterebbe sui nodi remoti che vogliono visualizzare l'andamento della gara. Infatti per via dei ritardi di rete il flusso dei dati non sarebbe stato omogeneo ed il tempo visualizzato non sarebbe stato solidale con quello reale.

5 Punti di divergenza

5.1 Prevedibilità della simulazione

La nostra volontà, già ripetuta più volte, era quella di avere una simulazione che fosse il più possibile (al 100% se escludiamo la caduta di un nodo remoto garantendo comunque la gestione di eventuali ritardi) prevedibile e calcolabile a priori da parte di un utente. Diciamo che si è considerata la simulazione come una funzione, la quale avrebbe dovuto dare (su richiesta) lo stesso output fissato l'input.

5.2 L'esperienza utente

L'avere una simulazione che rispecchi il passare del tempo sull'orologio dell'utente è una caratteristica che si può aggiungere ad un livello superiore dell'architettura del sistema. Deve essere fornita una simulazione di un sistema reale che produca dati che, una volta analizzati, risultino essere corretti. Per quanto riguarda la presentazione di tali dati ad un utente esterno riteniamo che sia un semplice problema di visualizzazione degli stessi. Si è pensato al simulatore più come uno strumento di analisi. La presentazione dei dati in modo 'realistico' è stato considerato quindi un requisito non mandatorio. Inoltre la nostra idea di soluzione porta a una correttezza di esecuzione anche velocizzata o senza delay, valorizzando così l'esperienza utente con velocizzazione e rallentamento della gara a livello di gui. Questo diverso punto di vista rispetto a quanto atteso per la produzione del prototipo ha portato alla prima divergenza, che poi, a cascata, ha influenzato ovviamente tutte le altre scelte progettuali e implementative, ed è questo il primo (e principale) punto di divergenza.

5.3 Distribuzione

Provando dopo aver compreso la soluzione proposta a pensare ad una eventuale distribuzione nella stessa ci siamo imbattuti in un secondo problema che nel nostro modo di procedere per lo sviluppo del progetto non ha intaccato il ragionamento. Secondo quanto visto studiando la soluzione proposta introdurre la distribuzione significa in parte introdurre delle componenti di non determinismo fuori dal nostro controllo. Questo è un ulteriore punto di divergenza sempre racchiuso nel primo dove la nostra volontà di garantire pieno determinismo e predicibilità ci ha portato verso una soluzione differente. Per mancata dimestichezza con la tecnologia, non si è riuscito a risolvere il problema con quanto offerto dal linguaggio. Questo ha portato in una direzione in cui, non riuscendo a rispecchiare una soluzione sulle primitive/strutture date, si sono dovute reinventare delle strutture che permettessero di risolvere il problema in base alla progettazione studiata. In questo modo si è potuto avere totale controllo / consapevolezza su quanto si stesse utlilizzando. Questo è un altro punto di divergenza, anche se di natura diverso dal precedente in quanto è dovuto non a una scelta progettuale ma a una mancanza di padronanza degli strumenti a disposizione.

5.4 Assunzioni

Altro punto di divergenza che ci ha portato a scelte progettuali diverse è dovuta al fatto che abbiamo volutamente evitato di fare assunzioni sull'ambiente

di esecuzione, chiedendo requisiti minimi (installazione di librerie come polyorb e xmlada, ambiente per eseguire ada e una installazione java) senza vincoli di processori o quant'altro. Quindi non sono stati effettuati calcoli riguardanti il numero (approssimato) di cicli di clock per l'esecuzione delle operazioni richieste alle entità attive. Tali operazioni non sono solamente quelle legate all'attraversamento dei tratti del circuito ma anche produzione di dati per la presentazione all'utente finale oltre che variazione a piacimento,e in modo scalabile, di un'eventuale intelligenza artificiale.

5.5 Conclusioni

Cercando di trarre le conclusioni per quanto riguarda i punti di divergenza ci si rende conto che la nostra idea iniziale di garantire (a nostro modo di vedere) la massima calcolabilità di quello che succede nel sistema (anche distribuendolo) è stata il punto principale di divergenza che ha poi portato a una serie di decisioni che ci hanno allontanato sempre di più dalla soluzione attesa. Le motivazioni che ci hanno spinto verso tali divergenza verranno analizzate in seguito, certo è che pensando fin da subito ad un sistema che funzionasse su sospensioni assolute tutte le altre scelte per risolvere i problemi sarebbero state una conseguenza e quindi saremmo arrivati alla soluzione attesa. Allo stesso modo le nostre scelte iniziali hanno condizionato tutto il progetto portandolo a divergere rispetto a quanto appreso durante il colloquio.

6 Analisi critica dei possibili errori (da parte nostra)

- worst-case troppo pessimistico (e che non si verificherà mai)
- visione un po' troppo java-oriented, ma è nel nostro bagaglio culturale.
- mancanza di padronanza degli strumenti per risolvere il problema (Ada)
- dato un problema difficile, senza una adeguata esperienza, di trovare necessariamente la soluzione ottima trovando comunque una corretta e funzionante
- la presentazione della simulazione non risulta essere scandita da dei tempi reali. L'utente che guardi il proprio orologio vedrà due flussi temporali diversi fra competizione e realtà. Sarebbe possibile tuttavia rielaborare i dati (forniti dal simulatore) in modo da poterli presentare associati ad un tempo realistico. Questo avrebbe tuttavia introdotto un secondo problema.

Ovvero l'utente non avrebbe potuto interagire sul sistema facendo affidamento sul tempo (far rientrare il concorrente al box ad un determinato istante t).

- pensiero di entità attiva indipendente dal linguaggio -¿ pensiero più java oriented? Si. La principale causa che ha portato alle divergenze viste ed analizzate in precedenza è sicuramente la nostra abitudine al paradigma di programmazione e al modello di concorrenza Java. Il ragionamento sviluppato era ovviamente worst-case driven. Nel nostro caso il caso pessimo era il più catastrofistico possibile senza assunzioni di controllo sulle entità attive. Presa coscienza della soluzione suggerita al colloquio ci rendiamo conto che usando il modello di concorrenza in Ada lo scenario worst-case risulta essere molto più blando rispetto a quello pensato in fase di progettazione. Dopo aver valutato la bozza di soluzione attesa sicuramente il worst-case reale si riduceva al risveglio di tutti le entità attive nel sistema nello stesso istante, fatto che nell'ambiente di esecuzione di Ada non influisce con ritardi significativi (infatti i tempi utilizzati nella simulazione sono millisecondi, che rispetto a una granularità al miliardesimo di secondo dell'ambiente di esecuzione portano a ritardi non visibili da un osservatore). All'inizio della fase di progettazione si era ipotizzato di affidarsi ad un tempo di riferimento assoluto. Dopo aver riscontrato i primi potenziali problemi nel sistema concorrente locale (valutazione del nostro worst-case) e costruiti possibili scenari distribuiti abbiamo riscontrato dal nostro punto di vista l'incontrollabilità del sistema se gestito con tempi assoluti. La soluzione progettata quindi utilizza tempi relativi, gestibili completamente da noi e portando a una simulazione completamente riproducibile e controllabile, anche con i delay e i problemi connessi alla rete. La base di tutti questi ragionamenti, che ci ha portato a commettere errori di valutazione che si sono poi propagati a cascata, è sicuramente l'assunzione di un worst-case che alla luce di quanto visto durante il colloquio orale è ben distante dalla realtà, in quanto non riproducibile nell'ambiente di esecuzione scelto. L'idea della nostra progettazione comunque era di mantenersi volutamente distanti dal linguaggio (e dall'ambiente scelto) per progettare il sistema e quindi si è arrivati a una reingegnerizzazione di strutture e meccanismi già presenti poi nell'ambiente scelto. La scelta di riprogettare alcuni meccanismi è dovuta quasi certamente al pensiero java oriented che abbiamo come bagaglio culturale. Inoltre certamente l'utilizzo dei tempi relativi gestiti a codice è stata influenzata dalla scarsa padronanza del paradigma di programmazione Ada. Probablimente questo utilizzo di tempi relativi ha compromesso la semplicità del prodotto finale

anche se riteniamo che il funzionamento sia corretto e verificabile.

- Diciamo quindi che la scelta del livello di non determinismo può avere influenzato, al pari della scelta di una progettazione totalmente 'language-indipendent', la direzione presa in fase di progettazione. Ci rendiamo conto che in una soluzione come quella suggerita i ritardi di rete avrebbero potuto essere 'mappati' su ritardi esistenti nella realtà. Ma questo non è stato un pensiero fatto in fase di analisi. Non si voleva progettare cercando di giustificare o associare il non-determinismo intrinseco dato dalla rete a un non-determinismo reale. Abbiamo voluto invece raggiungere il massimo determinismo possibile per poi aggiungere eventualmente in un secondo momento delle componenti che permettessero di fornire una fattore "random" alla gara.