Indice

Elenco delle figure

Elenco delle tabelle

Acronimi

SF Shadow Framework

Capitolo 1

Introduzione

In questo capitolo vengono presentati brevemente i contenuti dei diversi capitoli della presente tesi di laurea, vengono fornite una descrizione iniziale del progetto di tesi e dei suoi obbietti e presentate alcune considerazioni generali in merito al progetto stesso.

1.1 Lo Shadow Framework

1.2 Obbiettivo del progetto

L'obbiettivo del progetto di tesi nasce dall'idea di produrre un'applicazione dimostrativa delle funzionalit di rete offerte dallo Shadow Framework. L'applicazione che si voleva ottenere era una coppia client-server in cui il server fosse in grado di gestire connessioni simultanee da parte di un numero indefinito di client. Ogni client, ottenuta una connessione con il server, doveva assere in grado di visualizzare una scena iniziale navigabile, richiedendo solamente i dati relativi all'ambiente in prossimit di un eventuale avatar. Successivamente si voleva analizzare due possibili approcci: uno in cui, secondo le necessit, il client avrebbe richiesto al server i dati aggiuntivi riguardo la scena, ad esempio una volta raggiunti i bordi dell'ambiente, oppure un secondo in cui il server, comunicando attivamente con il client, tiene traccia degli spostamenti nella navigazione e fosse in grado di comporre in modo dinamico dei pacchetti di dati prevedendo le necessit del client.

Le astrazioni del layer dati del framework sono state progettate specificatamente per consentire lo sfruttamento della comunicazione di rete, ma fino a quel momento non era stata fatta alcuna specifica implementazione che la utilizzasse. Si desiderava perci produrre questo tipo di applicazione anche per individuare e correggere i probabili bug presenti nel codice e dovuti a vincoli di sincronizzazione ancora non affrontati dato che fino a quel momento tutti i test erano stati effettuati con dati sulla macchina locale.

L'obbiettivo della tesi cos diventato quello di produrre appunto un'implementazione delle astrazioni del framework che utilizzasse la comunicazione di rete per reperire i dati da visualizzare.

1.3 Considerazioni generali

1.4 Organizzazione della tesi

Capitolo 2

Lo Shadow Framework 2.0

Questo capitolo presenta inizialmente una panoramica sui moderni sistemi per la programmazione di grafica tridimensionale. Successivamente viene presentato lo ShadowFramework 2.0, le sue caratteristiche, la sua struttura e i suoi sviluppi futuri.

Capitolo 3

Gestione dei dati nello Shadow Framework 2.0

La gestione dei dati un compito molto importante all'interno del framework. Attraverso l'utilizzo di un layer di gestione dati astratto, ogni mudulo del framework pu essere salvato e caricato da file o trasferito attraverso un qualsiasi flusso di dati. In questo capitolo viene presentata l'astrazione utilizzata dallo Shadow Framework nella gestione dei dati, le funzionalit messe a disposizione ed i principali package e moduli coinvolti.

3.1 Il package `shadow.system.data`

Questo package contiene una serie di classi ed interfacce su cui si basa l'astrazione dei dati del framework.

3.1.1 SFDataObject

Uno dei moduli principali del package è `SFDataObject`, che rappresenta un'interfaccia con funzionalità di base comuni ad ogni oggetto che contiene dati. Ogni oggetto di questo tipo può perciò:

- essere scritto su di un `SFOutputStream`;
- essere letto da un `SFInputStream`;
- essere clonato;

I DataObject si basano sul *Composite Pattern*: possono essere semplici o contenere un insieme di oggetti figli, il fatto che sia gli oggetti complessi che gli oggetti semplici condividano la stessa interfaccia permette di trattare gli oggetti in modo uniforme. Un oggetto contenitore dovrà semplicemente richiamare lo stesso metodo di interfaccia per tutti gli oggetti figli i quali,

se oggetti semplici, hanno la responsabilità di implementare l'algoritmo per leggere o scrivere se stessi da uno stream.

Tutti i componenti SF utilizzano dei DataObject per incapsulare i dati in modo che questi ultimi possano essere letti e scritti utilizzando stream appropriati.

3.1.2 SFDataset

Un altro modulo importante per la gestione dei dati è `SFDataset`. Un Dataset è un oggetto che contiene un DataObject e informazioni sul proprio tipo, rappresentato tramite una stringa. Per rendere possibile il salvataggio e caricamento di collezioni di Dataset da file tramite input e output stream, è implementato nel framework un meccanismo di istanziazione di Dataset tramite una Factory. A loro volta i Dataset possono essere incapsulati in un DataObject usando un oggetto `SFDatasetObject`.

3.1.3 SFDataCenter

Il **DataCenter** é il nodo fondamentale della gestione dei dati all'interno del framework. É un oggetto *singleton* a cui le applicazioni accedono per richiedere i Dataset di cui hanno bisogno fornendo un'astrazione su come i dati sono effettivamente reperiti. Per poter funzionare, al DataCenter deve essere fornita un'implementazione per:

- `SFAbstractDatasetFactory`
- `SFIDataCenter`

L'implementazione di `SFAbstractDatasetFactory` deve essere essere una factory in grado di generare istanze di tutti i tipi di Dataset necessari all'applicazione, il package fornisce un'implementazione di default, di nome `SFGenericDatasetFactory`, che è sufficiente inizializzare passando un'istanza base dei Dataset.

L'interfaccia `SFIDataCenter` fornisce l'astrazione di una Mappa di Dataset identificati attraverso il proprio nome, attraverso di essa possiamo chiedere ad un oggetto che la implementa di recuperare un Dataset. Inoltre il Dataset recuperato non viene restituito direttamente, ma viene inviato attraverso un meccanismo di callback ad una implementazione di `SFDataCenterListener` passata come parametro, nel momento in cui è pronto. Questo tipo di astrazione permette di separare la logica di utilizzo del Dataset da quella di come esso viene reperito, consentendo ad una applicazione di usare dati locali o dati di rete semplicemente cambiando l'implementazione di `SFIDataCenter`.

3.1.4 SFObjectsLibrary

É usata per memorizzare un set di Dataset ed al suo interno ogni elemento è identificato tramite un nome univoco. Un `SFObjectsLibrary` è a sua volta un Dataset, così che un ObjectsLibrary possa essere contenuta in altre ObjectsLibrary. É possibile, ad esempio, utilizzare una ObjectLibrary all'interno di implementazione di `SFIDataCenter` per creare una mappa di Dataset necessari al funzionamento di un'applicazione.

3.1.5 SFLibraryreference

Un LibraryReference è un DataObject che può essere usato da qualsiasi componente per avere un riferimento ad un Dataset memorizzato in una libreria.

Capitolo 4

Il Progetto SF-Remote-Connection

In questo capitolo viene descritto il progetto sf-remote-connection, i moduli che lo compongono, le funzionalit offerte e i package java prodotti.

Il progetto SF-Remote-Connection una libreria di classi java ospitate, al momento della scrittura di questo documento, sul portale code.google.com per la condivisione di codice open source.

4.1 Moduli

La classi che compongono il progetto sono suddivise in una serie di package. Alcuni di questi sono pensati per rappresentare una possibile estensione a quelli forniti dal framework stesso e ne riproducono la struttura e le convenzioni sui nomi, gli altri invece affiancano o utilizzano il framework nella costruzione dell'applicazione. Questi package possono perci essere raggruppati in una serie di macro-moduli suddivisi per funzionalit e finalit:

1. **Base Communication**
2. **RemoteDataCenter Tool**
3. **Client**
4. **Test Client**
5. **Server**
6. **Test Server**

4.1.1 Base Communication

Questo modulo riunisce le classi che consentono la creazione e la gestione di connessioni TCP/IP tra applicazioni client/server. Ne fa parte anche

la classe di utilit `GenericCommunicator` che oltre a consentire la gestione della connessione assegnatagli la utilizza per fornire funzionalit di lettura e scrittura di messaggi testuali attraverso il canale aperto. Il modulo composto dal package `sfrc.base.communication` ed una libreria totalmente indipendente dal framework.

4.1.2 RemoteDataCenter Tool

Questo modulo raggruppa una serie di classi pensate per essere una estensione del framework e per essere utilizzate principalmente all'interno di una applicazione client. La sua funzione principale consiste nel fornire un ponte tra l'astrazione del reperimento dati fornita dal framework e il meccanismo di effettivo reperimento dei dati.

La classe chiave del modulo `SFRemoteDataCenter`: le richieste di Dataset effettuate al DataCenter vengono passate a questa classe che le esamina verificando che il dato richiesto sia presente nella libreria dell'applicazione. Se il Dataset non presente, al richiedente restituito un Dataset sostitutivo temporaneo scelto opportunamente, contemporaneamente viene generata una richiesta e aggiunta ad un buffer di richieste, questo pu essere utilizzato da un modulo esterno in grado di effettuare l'effettivo reperimento dei dati. Il modulo composto dai package `shadow.system.data.remote.wip`, `shadow.system.data.object.wip` e `shadow.renderer.viewer.wip`.

4.1.3 Client

Questo modulo raggruppa tutte quelle componenti generiche che possono essere utilizzate all'interno di una qualsiasi applicazione client e che servono ad implementare l'effettivo reperimento dei dati. Esso si pone al di sotto del modulo **RemoteDataCenter Tool** ed utilizza il modulo **Base Communication** per la gestione del canale di comunicazione e la sua implementazione pensata per il multi-threading. Il package che compone questo modulo **sfrc.application.client**.

4.1.4 Test Client

In questo modulo sono raccolte le implementazioni test

4.1.5 Server

4.1.6 Test Server

Capitolo 5

Test e Risultati

Dopo aver definito nel precedente capitolo la struttura del progetto, vengono qui presentati i test effettuati ed i risultati ottenuti.

Capitolo 6

Conclusioni

Capitolo 7

SFRemoteConnection

Appunti sparsi sul progetto

7.1 Obbiettivo del progetto

L'obbiettivo del progetto di tesi nasce dall'idea di produrre un'applicazione dimostrativa della capacità del framework di funzionare con dati residenti su di una macchina remota raggiungibile tramite rete. L'applicazione che si voleva ottenere era una coppia client-server in cui il server fosse in grado di gestire connessioni simultanee da parte di un numero indefinito di client. Ogni client, ottenuta una connessione con il server, doveva assere in grado di visualizzare una scena iniziale navigabile, richiedendo solamente i dati relativi all'ambiente in prossimità di un eventuale avatar. Successivamente si voleva analizzare due possibili approcci: uno in cui, secondo le necessità, il client avrebbe richiesto al server i dati aggiuntivi riguardo la scena, ad esempio una volta raggiunti i bordi dell'ambiente, oppure un secondo in cui il server, comunicando attivamente con il client, tiene traccia degli spostamenti nella navigazione e fosse in grado di comporre in modo dinamico dei pacchetti di dati prevedendo le necessità del client.

Le astrazioni del layer dati del framework sono state progettate specificatamente per consentire lo sfruttamento della comunicazione di rete, ma fino a quel momento non era stata fatta alcuna specifica implementazione che la utilizzasse. Si desiderava perciò produrre questo tipo di applicazione anche per individuare e correggere i probabili bug presenti nel codice e dovuti a vincoli di sincronizzazione ancora non affrontati dato che fino a quel momento tutti i test erano stati effettuati con dati sulla macchina locale.

L'obbiettivo della tesi è così diventato quello di produrre appunto un'implementazione delle astrazioni del framework che utilizzasse la comunicazione di rete per reperire i dati da visualizzare.

7.1.1 Infrastruttura di rete

Date le specifiche iniziali è stato necessario stabilire quali componenti fosse necessario sviluppare ed in quale ordine. Come prima cosa si è stabilito di sviluppare una serie di test che replicassero quelli già funzionanti in locale, per poter fare questo era necessario creare innanzitutto una infrastruttura per la comunicazione di rete tra client e server.

Se da lato client è ovvia la necessità di sviluppare uno strato dell'applicazione che si occupasse della comunicazione di rete, da lato server si presentano diverse possibilità:

1. utilizzare un file server che permettesse semplicemente di accedere ai file contenti i dati tramite la rete;

2. utilizzare un application server java, come Tomcat o Glassfish, a cui un'applicazione client potesse connettersi e che attraverso l'esecuzione di servlet realizzasse il trasferimento dei dati da server a client;

3. utilizzare un'applicazione server ad hoc appositamente sviluppata;

La prima soluzione è probabilmente la più semplice, ma la meno flessibile dato che consente solo di un accesso diretto ai file di descrizione dei dati senza alcuna possibilità di un'elaborazione lato server e spostando tutto il peso di un'eventuale interazione tra client direttamente su quest'ultimi.

La seconda soluzione offre più possibilità e flessibilità rispetto alla prima: l'utilizzo di un application server java mette a disposizione una piattaforma che consente una pre-elaborazione dei dati lato server e che possiede direttamente una serie di componenti per la gestione di compiti complessi legati alle sessioni degli utenti, come ad esempio l'autenticazione e la sicurezza. Nonostante i pregi, questo tipo di soluzione possiede anche lati negativi: gli application server generici non sono sviluppati per questo tipo di applicazioni e non era garantita una flessibilità sufficiente per cui all'aumentare della complessità del progetto e delle sue esigenze non fosse necessario abbandonare l'architettura.

La terza soluzione è sicuramente la più flessibile ed estendibile dato che, se necessario, consente di modificare direttamente il server per adattarsi alle esigenze dell'applicazione. Lo svantaggio è la necessità di dover implementare da zero tutte quelle funzionalità non solo di comunicazione, ma anche di autenticazione o di sicurezza che una soluzione già pronta potrebbe possedere nativamente.

Nella scelta tra le tre soluzioni hanno pesato prevalentemente la volontà di realizzare un'architettura funzionante con la scrittura di meno codice possibile e quella di mantenere una bassa complessità iniziale che non penalizzasse però un'estensione futura delle funzionalità. In quest'ottica la prima soluzione stata anche la prima ad essere scartata, perch sebbene garantisse un tempo di messa in opera molto basso non consente un'estensione di

funzionalit. La scelta finale ricaduta sulla terza soluzione che pur costringendo a rinunciare alle funzionalit avanzate della seconda, elimina difficolt e tempistiche di una installazione e configurazione dell'application server. Inoltre, limitando inizialmente lo sviluppo a funzionalit di base, possibile limitare la complessit del codice ad un livello non molto pi elevato rispetto alla seconda soluzione.

7.2 SFRemoteConnection

un'implementazione di `SFIDataCenter` che reperisse i dati attraverso la rete.

7.3

Capitolo 8

TMP

Pluto

8.1 tmp1

Topolino

8.1.1 tmp2

Paperino

tmp3

Qui

tmp4 Quo

tmp5 Qua