Indice

Elenco delle figure

Elenco delle tabelle

Acronimi

SF Shadow Framework

GPU Graphic Processing Unit

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

Questa tesi stata realizzata nell'ambito dello sviluppo dello Shadow Framework 2.0, progetto ideato e sviluppato dall'ingegner Alessandro Martinelli. Lo Shadow Framework (SF) un framework per lo sviluppo di applicazioni che fanno uso di grafica tridimensionale sia in ambito scientifico che per l'intrattenimento. Tra le sue caratteristiche principali vi sono la portabilit, la possibilit di lavorare sia con dati locali che con dati in remoto, la capacit di sfruttare in maniera estesa le caratteristiche delle moderne Graphic Processing Unit (GPU).[1]

Una descrizione pi dettagliata delle caratteristiche del framework viene fornita nel capitolo 2.

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.

[1] Una GPU un microprocessore dedicato alla generazione delle immagini visualizzate sullo schermo di un dispositivo, alleggerendo da questo carico il processore principale.

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 L'astrazione della gestione dati

All'interno di un'applicazione SF l'unit base di dati pu essere identificata con quello che viene definito SFDataset (3.2.2). Con Dataset si identifica quasi ogni tipo di dato, sia grafico che non, utilizzato all'interno del framework. La gestione dei dataset viene effettuata mediante un meccanismo centralizzato: ogni applicazione in esecuzione possiede un'istanza di SFDataCenter, questa classe un oggetto *Singleton*[1] a cui ogni componente pu accede per richiedere operazioni sui Dataset di interesse, operazioni che possono essere la lettura o la scrittura da uno stream specifico, la richiesta di una particolare istanza di un Dataset, identificata per nome, o la richiesta di una nuova istanza di Dataset, identificata per tipo. L'oggetto Singleton espone queste funzionalit traducendole internamente con chiamate ad una

[1] Con *Singleton* si intende il design pattern omonimo descritto in maniera pi accurata nella sezione B.1.2

factory concreta[2] di Dataset e ad una istanza dell'interfaccia SFIDataCenter, creando un'astrazione su come i Dataset siano effettivamente costruiti e reperiti. La factory concreta deve essere un'implementazione dell'interfaccia SFAbstractDatasetFactory in grado di istanziare, leggere o scrivere ogni tipo di Dataset utilizzato dall'applicazione. L'istanza dell'interfaccia SFIDataCenter tiene traccia dei Dataset istanziati con nome, restituendone un riferimento a chi ne fa richiesta attraverso la chiamata a funzioni di callback.

3.2 Il package `shadow.system.data`

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

3.2.1 SFDataObject

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

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

I DataObject si basano sul *Composite Pattern*B.2.1: 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 dovr semplicemente richiamare lo stesso metodo di interfaccia per tutti gli oggetti figli i quali, se oggetti semplici, hanno la responsabilit 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.

[2] Si fa riferimento al pattern di programmazione *Abstract Factory* descritto nella sezione B.1.1

3.2.2 SFDataset

Un altro modulo importante per la gestione dei dati `SFDataset`. Un Dataset un oggetto che contiene un DataObject e informazioni sul proprio tipo, rappresentato tramite una stringa. L'interfaccia SFDataset definisce un'interfaccia per oggetti di questo tipo, la quale consente di accedere al nome del tipo specifico, al DataObject contenuto e di creare una nuova istanza delle stesso tipo. A loro volta i Dataset possono essere incapsulati in un DataObject usando un oggetto `SFDatasetObject`.

3.2.3 SFAbstractDatasetFactory

Questa interfaccia definisce le operazioni base richieste ad una DatasetFactory, queste operazioni consistono in:

- lettura/scrittura di un Dataset da uno stream
- la creazione di una nuova istanza di un Dataset specificato per tipo

3.2.4 SFGenericDatasetFactory

Consiste in una implementazione concreta di default dell'interfaccia `SFAbstractDatasetFactory`. Per consentire il riutilizzo del codice stata resa configurabile: l'aggiunta di un metodo addSFDataset() consente di generare, in un oggetto GenericDatasetFactory, un elenco di Dataset istanziabili. Quando verranno chiamati i metodi dell'interfaccia SFAbstractDatasetFactory sull' oggetto GenericDatasetFactory diviene sufficiente richiamare il metodo opportuno del Dataset del tipo richiesto o del DataObject in esso contenuto.

3.2.5 SFIDataCenter

L'interfaccia `SFIDataCenter` fornisce l'astrazione di una Mappa di Dataset identificati attraverso il proprio nome, attraverso di essa possiamo chiedere di recuperare un Dataset ad un oggetto che la implementa. Quest'oggetto non deve restituire direttamente il Dataset recuperato, ma deve farlo attraverso un meccanismo di callback ad una implementazione dell'interfaccia `SFDataCenterListener` passata come parametro, nel momento in cui il dato disponibile.

3.2.6 SFDataCenterListener

Questa interfaccia definisce la callback che un componente deve implementare per effettuare una richiesta al DataCenter. Questa callback viene richiamata quando il Dataset richiesto pronto.

3.2.7 SFDataCenter

Il **DataCenter** il nodo fondamentale della gestione dei dati all'interno del framework.

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`

Come precedentemente esposto, l'implementazione di `SFAbstractDatasetFactory` deve essere essere una factory in grado di generare istanze di tutti i tipi di Dataset necessari all'applicazione.

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`.

3.2.8 SFObjectsLibrary

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

3.2.9 SFLibraryreference

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

3.2.10 SFInputputStream e SFOutputStream

Queste interfacce definiscono le operazioni necessarie che uno stream di input o di output deve implementare affinch sia possibile leggere o scrivere su di esso dei DataObject.

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.

4.1 Informazioni generali

In base agli obbiettivi di progetto esposti al paragrafo 1.2, ci che stato prodotto consiste in una estensione dello Shadow Framework e all'implementazione di un sistema di comunicazione dati sfruttato da questa estensione. 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. Il codice attualmente rilasciato sotto licenza GNU GPL v3. Si rimanda all'appendice A per le versioni delle librerie utilizzate.

4.2 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 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**

4.2.1 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 dai package `sfrc.base.communication` e `sfrc.base.communication.s`

4.2.2 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 meccanismo dei Dataset sostitutivi viene descritto esaustivamente nella sezione 4.3.

Il modulo composto dai package `shadow.system.data.remote.wip`, `shadow.system.data.object.wip` e `shadow.renderer.viewer.wip`.

4.2.3 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**.

4.2.4 Test Client

In questo modulo sono raccolte le implementazioni delle applicazioni di test per le componenti lato client. Questi test sono stati creati principalmente per riprodurre quelli gi presenti nel progetto SF20LiteTestWorld e usati per mostrare le capacit del framework.

Il package che compone questo modulo **sfrc.application.client.test**.

4.2.5 Server

Similmente al modulo per le componenti client, in questo vengono raggruppate delle componenti generiche utili alla realizzazione di una applicazione server. Queste componenti si pongono da tramite tra l'applicazione e il modulo di **Base Communication** tramite cui realizzano l'effetivo trasferimento dei Dataset verso il client connesso. Anche in questo caso l'implementazione pensata per il funzionamento multi-thread in parallelo con l'applicazione principale che pu cos gestire pi client connessi contemporaneamente ed eseguire altre operazioni. Vengono fornite infine anche delle interfacce utili per effettuare l'inizializzazione dei dati e per configurare il protocollo di comunicazione.

Il modulo composto dal package `sfrc.application.server`

4.2.6 Test Server

Di questo modulo fanno parte le implementazioni di server e applicazioni di test utilizzati per verificare il comportamento delle componenti lato server. Oltre a queste vengono usati per testare i client in differenti condizioni di comunicazione simulate dai server, come ad esempio una risposta a singhiozzo, ecc.

Di questo modulo fanno parte i package `sfrc.application.server.test` e `sfrc.application.server.task`.

4.3 Dataset sostitutivi

Per evitare che, in seguito alla richiesta di un Dataset non presente nella libreria locale di un'applicazione, i moduli richiedenti rimanessero in attesa dei dati bloccando di fatto l'esecuzione, stato deciso di realizzare un meccanismo di sostituzione-aggiornamento delle richieste. Successivamente ad una richiesta il RemoteDataCenter restituisce, attraverso la callback del richiedente, un Dataset sostitutivo temporaneo di tipo compatibile a quello richiesto. Contemporaneamente viene generata una richiesta remota che attende di essere evasa. Quando i dati effettivi arrivano dalla rete viene eseguito un update del dato richiamando nuovamente la callback del richiedente. Dato che pi moduli dell'applicazione potrebbero fare richiesta dello stesso Dataset, tutte le callback dei richiedenti vengono memorizzate e, al momento dell'update, richiamate in successione. Per permettere il funzionamento di questo automatismo si resa necessaria la realizzazione di un nuovo tipo di Dataset, l'SFDatasetReplacement, e di una libreria di Dataset sostitutivi.

Utilizzato all'interno di una ObjectsLibrary un DatasetReplacement permette di realizzare una lista di sostituzione che associa il nome di un Dataset Alfa richiesto, a quello di un Dataset sostitutivo di default Beta e ad un timestamp. L'associazione tra nomi viene usata per una ricerca diretta del dato sostitutivo da utilizzare, mentre il timestamp stato introdotto per lo sviluppo futuro di logiche di aggiornamento della lista di sostituzione. Le liste di sostituzione sono state inoltre rese configurabili attraverso file XML leggibili dal decoder interno del framework.

La libreria dei Dataset di default stata invece realizzata progressivamente durante l'implementazione dei test, quando si rendeva necessaria la costruzione di un Dataset specifico dato che quelli gi realizzati erano di tipo incompatibile.

L'utilizzo di questo meccanismo richiede necessariamente una fase di inizializzazione in cui viene fatto il download o il caricamento da disco locale della lista di sostituzione e della libreria dei Dataset di default. Avendo la possibilit di estendere il protocollo di comunicazione in modo da rendere

espandibili queste due librerie durante l'esecuzione, possibile mantenere la loro dimensione iniziale contenuta.

4.4 I Package java

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

Appendice A

Note sul Software

A.1 Versioni dei software utilizzati

Appendice B

Design Pattern

Nel campo dell'Ingegneria del Software con Design Pattern si intende uno schema di progettazione del software utilizzato per risolvere un problema ricorrente. Molti di questi schemi sono stai pensati per il paradigma di programmazione ad oggetti e descrivono, utilizzando ereditariet e interfacce, le interazioni e relazioni tra oggetti e classi.

In questa appendice vengono esposte alcune note generali sui design pattern citati nel testo, sia che siano stati implementati direttamente o semplicemente coinvolti nello sviluppo del progetto di tesi perch sfruttati dalle parti interessate del framework. Nella trattazione viene descritto l'obbiettivo di ogni schema, l'utilit, note implementative e alcuni benefici della sua applicazione, vengono inoltre raggruppati per categoria cos come vengono presentati in [1].

B.1 Pattern Creazionali

I pattern creazionali sono usati per creare un'astrazione sul processo di instanziazione degli oggetti, in modo da rendere il sistema indipendente da come gli oggetti vengono effettivamente creati. Questo tipo di pattern diviene importante quando un sistema dipende principalmente da oggetti composti da aggregati di componenti pi piccole, rispetto ad oggetti gerarchici organizzati in base all'ereditariet.

B.1.1 Abstract Factory

L'obbiettivo di questo pattern fornire un'interfaccia per la creazione di famiglie di oggetti imparentati o dipendenti tra loro, eliminado la necessit di specificare i nomi delle classi concrete all'interno del codice.

In questo modo un sistema pu essere indipendente da come vengono creati gli oggetti concreti e pu essere configurato per utilizzare diverse famiglie di oggetti a seconda delle necessit. Un caso esemplificativo quello di un tool per l'implementazione di interfacce grafiche che supporta diversi look-and-feel. L'utilizzo di un diverso look-and-feel comporta la definizione di una nuova famiglia di componenti grafiche le cui caratteristiche base sono per fondamentalmente identiche: un pulsante pu essere premuto e disegnato a schermo. L'applicazione dovrebbe essere appunto indipendente da come il pulsante viene istanziato, ma essere in grado di riconoscerlo per le sue funzionalit.

Per ottenere ci possibile definire una classe astratta o un'interfaccia differente per ogni componente generico desiderato. Si definisce poi una classe Factory astratta la cui funzione sia quella di creare istanze delle componenti astratte generiche. Se per ogni famiglia che si desidera implementare si generano sottoclassi concrete per ogni componente astratta e si definisce una Factory in grado di istanziarle, l'applicazione potr utilizzare la Factory concreta che preferisce in modo del tutto indipendente da quale famiglia essa gestisce.

Questo schema permette di cambiare le famiglie di componenti in modo semplice e consente di isolare le classi concrete, dato che le componenti vengono utilizzate attraverso la loro interfaccia.

B.1.2 Singleton

Lo scopo di questo pattern garantire che di una determinata classe possa essere creata una unica istanza, fornendone un singolo punto di accesso globale.

L'utilit di questo pattern dovuta al fatto che spesso, all'interno di un'applicazione, ci sono servizi e funzionalit condivise a cui deve essere associata una istanza univoca. Esempi classici in proposito sono il gestore del File System o il gestore dei servizi di stampa.

Per ottenere questa propriet spesso si delega alla classe stessa la responsabilit di controllare la creazione delle propria istanza rendendo privato il suo costruttore, impedendo di fatto l'istanziazione dall'esterno.

Utilizzare questo schema consente di avere un pi stretto controllo degli accessi alla risorsa, inoltre estendendo la classe possibile raffinare le funzionalit della risorsa e configurare l'applicazione a runtime per utilizzare l'istanza che serve per un caso specifico.

B.2 Pattern Strutturali

Questo tipo di pattern si occupano di come classi e oggetti vengono composte e interagiscono nella formazione di strutture pi complesse. Un obbiettivo di questi pattern permettere la composizione di oggetti complessi che uniscano le funzionalit di moduli pi piccoli rendendo quest'ultimi il pi possibile riutilizzabili.

B.2.1 Composite

Lo scopo di questo pattern consentire una gestione uniforme tra oggetti semplici ed oggetti compositi. Questo si traduce in una organizzazione degli oggetti in una struttura ad albero in cui ogni nodo un oggetto aggregato e ogni foglia un oggetto semplice.

Spesso si desidera poter trattare oggetti composti nello stesso modo in cui gestiamo le sue componenti ignorando come siano effettivamente implementate le loro funzionalit, un esempio significativo sono le interfacce grafiche in cui comodo poter gestire gruppi di componenti, come pulsanti o etichette, in modo unitario come se si trattasse di una componente singola.

Una metodologia per ottenere questo tipo di interazione l'utilizzo di una classe astratta che rappresenti sia gli oggetti semplici, o primitive, sia gli oggetti composti, o contenitori.

L'utilizzo di questo pattern consente di rendere meno complessi i moduli che utilizzano le strutture composite, che non hanno bisogno di sapere se stiano trattando una foglia o un nodo dell'albero. Consente inoltre di aggiungere in modo semplice nuovi componenti senza laboriosi interventi sul codice pre-esistente.

Bibliografia

[1] Ralph Johnson John Vlissides Erich Gamma, Richard Helm. *Design Patterns: Elements of Reusable Object-Oriented Software*. AddisonWesley Professional, 1994.