Indice

Elenco delle figure

Elenco delle tabelle

Acronimi

SF Shadow Framework

GPU Graphic Processing Unit

UML Unified Modeling Language

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 dovuti a vincoli di sincronizzazione non evidenziati dai test effettuati con dati sulla macchina locale.

L'obbiettivo della tesi cos diventato quello di produrre dei moduli di libreria che fossero utili allo sviluppo di applicazioni client-server.

La progettazione di questi moduli si focalizzata su alcuni punti cardine che rappresentano la chiave dell'aspetto di sviluppo legato al progetto. Dato che la struttura del framework ideata con l'obbiettivo di essere fortemente estendibile, si voleva che i moduli utilizzassero i meccanismi e le astrazioni previste. Si desiderava inoltre che i moduli fossero a loro volta progettati per l'estendibilit e il riutilizzo del codice.

1.3 Considerazioni generali

MENZIONARE I TEST

Qui si pu fare un discorso sulle parti del framework su cui necessario focalizzarsi facendo riferimenti ai capitoli 2 e 3.

1.4 Organizzazione del documento

Il presente documento organizzato secondo la seguente suddivisione in capitoli:

- **Capitolo 2:** in cui viene presentata una panoramica generale sullo Shadow Framework 2.0 in rapporto al panorama generale sui framework di programmazione di grafica tridimensionale real-time.
- **Capitolo 3:** in descritta l'astrazione di gestione dei dati interna al framework e come essa viene utilizzata dalle applicazioni SF.
- **Capitolo 4:** in cui viene descritto il progetto Sf-Remote-Connection, i moduli che lo compongono, le funzionalit offerte e i package java prodotti.
- **Capitolo 5:**
- **Capitolo 6:**

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 (3.2.3). 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* che realizza un *Bridge* tra l'astrazione di reperimento dati e la sua implementazione concreta[1]. Ogni componente pu accedere al DataCenter 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,

[1] Con *Singleton* e *Bridge* si intendono i design pattern omonimi descritti pi in dettaglio nell'appendice B

identificata per tipo. L'oggetto Singleton espone queste funzionalit traducendole internamente con chiamate ad una 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 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.

3.2.2 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.2.2: 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

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

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.

3.2.3 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`.

3.2.4 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

3.2.5 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.

3.2.6 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.

3.2.7 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.

3.2.8 SFDataCenter

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

un oggetto *Singleton* (B.1.2) a cui le applicazioni accedono per richiedere i Dataset di cui hanno bisogno. Questa classe utilizza anche il pattern *Bridge* (B.2.1) per 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`.

3.2.9 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.

3.2.10 SFLibraryreference

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

Capitolo 4

Il Progetto SF-Remote-Connection

Vengono ora presentati i moduli software realizzati nel corso dello sviluppo del progetto di tesi. Per meglio comprendere gli aspetti di sviluppo legati al progetto importante tener conto degli obbiettivi di progetto esposti al paragrafo 1.2 e della descrizione dei meccanismi di gestione dati interni del framework, descritti nel capitolo 3.

 giusto sottolineare che l'obbiettivo principale stato la produzione di librerie e tool per lo sviluppo di applicazioni, focalizzandosi sull'estendibilit e il riutilizzo del codice. Non a caso sono infatti presenti alcuni riferimenti di appendice ad alcuni design pattern particolarmente significativi nella produzione di questo tipo di softwaree e che sono utilizzati sia dal framework che dai moduli stessi.

Per il processo di sviluppo stata di fondamentale importanza la produzione parallela di una serie di test, presentati nel capitolo 5. Infatti la progettazione e la realizzazione dei moduli e dei meccanismi presentati in questo capitolo non stata svolta in maniera distinta, ma si trattato di un processo iterativo in cui i test hanno svolto pi di una volta un ruolo di guida nel refactoring del codice.

Il capitolo strutturato in modo da fornire innanzitutto una suddivisione e una descrizione dei moduli, vengono poi descritti il meccanismo dei Dataset sostitutivi e il protocollo di comunicazione. Infine presente una panoramica sui package java e le principali classi di ognuno di essi.

Si rimanda all'appendice A informazioni sul codice sorgente relativo al

progetto e per informazioni sulle versioni delle librerie utilizzate.

4.1 Moduli

La libreria di classi realizzata pu essere suddivisa in quattro macro-moduli suddivisi in base alle finalit e alle funzionalit. Di seguito viene data una descrizione degli stessi in questo ordine:

1. **Base Communication**
2. **RemoteDataCenter Tool**
3. **Client**
4. **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 dai package `sfrc.base.communication` e `sfrc.base.communication.sfutil`.

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 uno strato di comunicazione tra l'astrazione del reperimento dati fornita dal framework e il meccanismo di effettivo reperimento dei dati.

La classe chiave del modulo `SFRemoteDataCenter`: questa una classe di implementazione utilizzabile nel *Bridge* realizzato da `SFDataCenter`[1]. Le richieste di Dataset effettuate al DataCenter vengono passate a questa classe

[1] Per la classe `SFDataCenter` si rimanda al paragrafo 3.2.8 mentre per il pattern *Bridge* al paragrafo B.2.1.

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 4.2.

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 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 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 dei Dataset con successivo update 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.

4.3 Infrastruttura di rete

Date le specifiche iniziali esposte nel capitolo 1, una parte fondamentale del progetto stata decidere quale infrastruttura per la comunicazione di rete sfruttare per poter utilizzare e testare il modulo **RemoteDataCenter Tool**, che si occupa della gestione delle richieste di Dataset.

Se da lato client ovvia la necessit di sviluppare uno strato dell'applicazione che si occupa della comunicazione di rete, da lato server si sono presentate diverse possibilit:

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

La seconda soluzione offre pi possibilit e flessibilit 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 flessibilit sufficiente per cui all'aumentare della complessit del progetto e delle sue esigenze non fosse necessario abbandonare l'architettura.

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

Nella scelta tra le tre soluzioni hanno pesato prevalentemente la volont di realizzare un'architettura funzionante con la scrittura di meno codice possibile e quella di mantenere una bassa complessit iniziale che non penalizzasse per un'estensione futura delle funzionalit. 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.

4.3.1 Protocollo di comunicazione

Una volta stabilito di utilizzare un server sviluppato appositamente stato necessario stabilire le modalit di comunicazione tra client e server.

La comunicazione tra le applicazioni viene effettuata attraverso un protocollo basato su messaggi testuali che si colloca idealmente a livello di Sessione nel modello ISO/OSI [2] mentre a livello inferiore viene utilizzato il protocollo TCP/IP. L'utilizzo del TCP in quanto protocollo confermato giustificato in quanto consente di garantire l'arrivo dei messaggi di comunicazione. Questi messaggi sono principalmente dedicati allo scambio dei dati grafici, per cui la garanzia di arrivo e l'integrit dei dati sono prioritarie rispetto alle prestazioni.

I messaggi di comunicazione finora previsti sono strutturati secondo la seguente forma:

```
etichetta_messaggio[:dati:opzionali:...]
```

in cui `etichetta_messaggio` identifica il tipo di messaggio, mentre i dati opzionali dipendono dal messaggio stesso e sono divisi dall'etichetta e tra loro dal carattere " : ".

La gestione della comunicazione viene effettuata tramite una macchina a stati specifica per client e server, le quali si occupano di leggere ed effettuare l'analisi logico-sintattica dei messaggi generando le eventuali risposte o semplicemente modificando lo stato interno della macchina stessa.

Usare messaggi testuali per la comunicazione consente di semplificarne la gestione e rende possibile espandere il linguaggio semplicemente utilizzando nuove etichette, consentire tutto ci ha per reso necessario strutturare l'implementazione delle macchine a stati in modo che fossero altrettanto espandibili.

I messaggi attualmente implementati sono:

4.4 I Package java

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 sono librerie che affiancano il framework nella costruzione dell'applicazione.

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.

5.1 Test Client

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. Ne fanno parte anche le classi che implementano i task del protocollo di comunicazione lato client, per la cui trattazione fare riferimento alla sezione 4.3.1.

Il package che compone questo modulo `sfrc.application.client.test` e e `sfrc.application.client.task`.

5.2 Test Server

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. Ne fanno parte anche le classi che implementano i task del

protocollo di comunicazione lato server, fare riferimento alla sezione 4.3.1 per una trattazione pi approfondita.

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

Capitolo 6

Conclusioni

Appendice A

Note sul Software

A.1 Codice sorgente del progetto SF-Remote-Connection

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.

A.2 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 con schema Unified Modeling Language (UML) annesso 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.

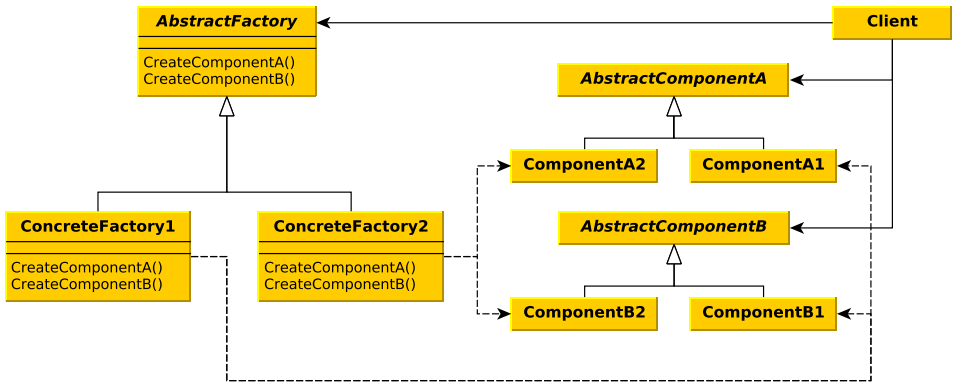

Figura B.1: Struttura UML del pattern Abstract Factory.

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, come `AbstractComponentA` e `AbstractComponentB` nella figura B.1. Si definisce poi una classe `AbstractFactory` 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 concreta in grado di istanziarle, come `ComponentA1` e `ConcreteFactory1` nell'esempio raffigurato, l'applicazione potr utilizzare la Factory concreta che preferisce in modo del tutto indipendente da quale famiglia essa gestisce.

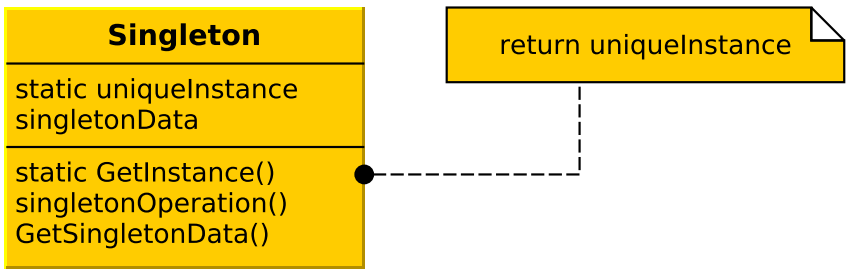

Figura B.2: Struttura UML del pattern Singleton.

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. In figura B.2 possiamo vedere lo schema UML di una classe che utilizza il pattern.

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 occupa 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 Bridge

Questo pattern ha l'intento di disaccoppiare un'astrazione dalla sua implementazione in modo che entrambe possano evolversi in maniera indipendente.

Ci particolarmente utile quando si desidera evitare un legame permanente tra astrazione e implementazione, e si vuole permettere di scegliere o cambiare quest'ultima durante l'esecuzione. utile inoltre quando sia astrazione che implementazione hanno la necessit di essere estendibili attraverso sottoclassi.

Un utile esempio per illustrare questo pattern prendere in esame un ipotetico toolkit per interfacce grafiche, per renderlo il pi possibile portabile su piattaforme diverse esso deve utilizzare un'astrazione che descriva le finestre in modo pi generico possibile. Se per ottenere queste propriet usassimo semplicemente una classe astratta `Window` da cui, usando l'ereditariet, costruire sottoclassi specifiche per ogni sistema da supportare, otterremmo due svantaggi:

1. Nell'evenienza di voler estendere la classe `Window` per coprire l'astrazione di altri tipi di finestra grafica esistente, per poter supportare tutte le piattaforme per cui esisteva una implementazione di `Window` dovremmo creare una sottoclasse specifica per ognuna di esse.

2. Il codice del client diventa dipendente dalla piattaforma. Quando vi la necessit di creare una finestra, deve essere istanziata una classe concreta specifica e questo lega fortemente l'astrazione con l'implementazione utilizzata rendendo pi complesso portare il codice del client su altre piattaforme.

Per eludere questi problemi il pattern Bridge separa la gerarchia delle classi che appartengono all'astrazione da quella delle classi di implementazione. In cima alle due gerarchie vi sono le classi `Window` e `WindowImplementation`, che compongono il Bridge vero e proprio. La prima rappresenta l'astrazione della finestra che verr utilizzata dal client, la seconda invece l'interfaccia

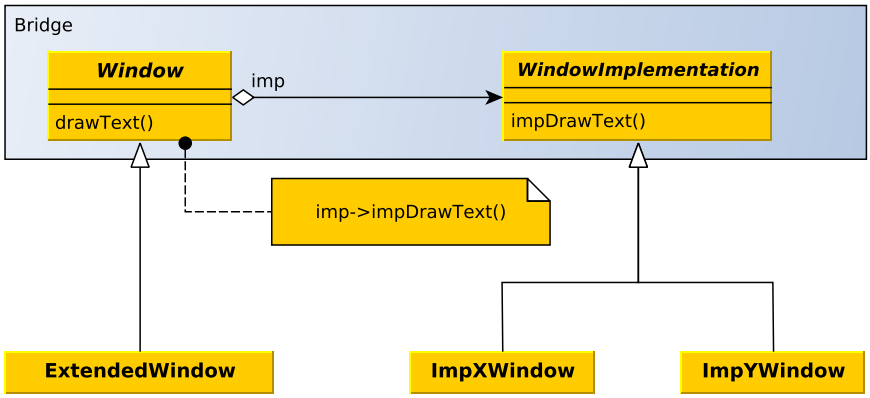

Figura B.3: Esempio di schema gerarchico dovuto all'applicazione del pattern Bridge.

che un'implementazione concreta deve utilizzare per essere utilizzata dall'astrazione. Come si pu vedere nella figura B.3, la classe `Window` rimappa i propri metodi su quelli dell'interfaccia `WindowImplementation`, o con una combinazione di essi. Questi vengono chiamati su di una implementazione concreta di `WindowImplementation` di cui `Window` possiede un riferimento `imp`.

L'utilizzo di questo schema permette di nascondere i dettagli implementativi ai client e di non avere effetti diretti su di esse quando l'implementazione cambia cos che il codice dei client non non ha un'effettiva necessit di essere ricompilato.

B.2.2 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. Il nodo composito avr al suo interno riferimenti ad oggetti figli che a loro volta possono essere semplici o compositi.

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

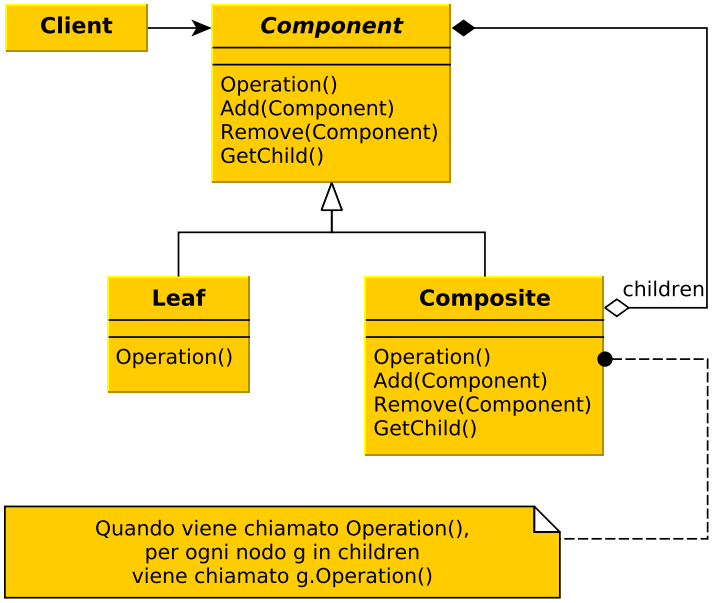

Figura B.4: Struttura UML del pattern Composite.

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. Nello schema UML in figura B.4, la classe astratta `Component` rappresenta sia oggetti foglia `Leaf` che nodi `Composite`. Un client pu richiamare indifferentemente i metodi di interfaccia di un oggetto sottoclasse di `Component` sia che si tratti di un una foglia o di un nodo. Nel caso di una foglia il metodo risponder direttamente in maniera appropriata mentre nel caso di un nodo verr richiamato lo stesso metodo per tutti gli oggetti figli.

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.

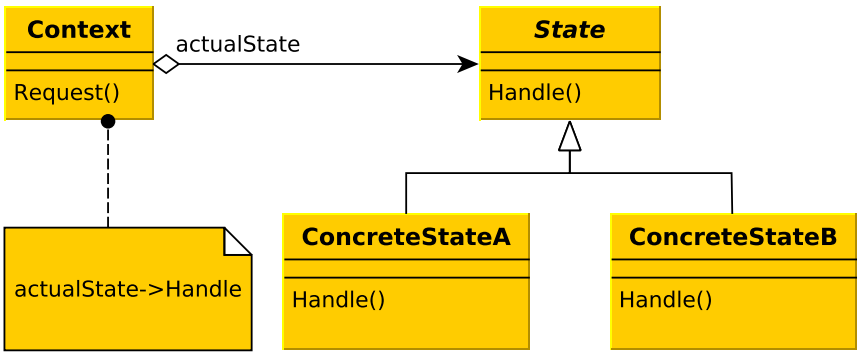

Figura B.5: Schema UML del pattern State.

B.3 Pattern Comportamentali

I pattern comportamentali riguardano gli algoritmi e la suddivisione delle responsabilit tra gli oggetti, per fare ci non si limitano solamente a descrivere i rapporti tra oggetti e classi, ma anche attraverso schemi di comunicazione tra di essi. Questi schemi permettono di descrivere complessi flussi di controllo spostando l'attenzione da essi alle connessioni fra gli oggetti.

B.3.1 State

Il pattern State ha come scopo quello di consentire ad un oggetto di cambiare il proprio comportamento durante l'esecuzione, in base alle variazioni del proprio stato interno.

Esempi di questo tipo di necessit possono essere trovati in oggetti che gestiscono connessioni e che devono rispondere in modo diverso in base allo stato della connessione stessa, oppure nella costruzione di macchine a stati che implementano algoritmi di controllo.

La figura B.5 mostra in uno schema UML la struttura del pattern. La classe `Context` rappresenta l'interfaccia che l'oggetto a stato variabile presenta verso i client. Al suo interno mantiene una istanza (`actualState`) di una sottoclasse concreta della classe `State`. La classe astratta `State` definisce un'interfaccia per incapsulare il comportamento associato ad uno stato. Le sottoclassi concrete di `State` sono associate ad un effettivo stato e implementano il comportamento che l'oggetto `Context` assume in quel caso.

Quando i client effettuano richieste all'oggetto `Context` attraverso la sua interfaccia, viene richiamato il metodo di interfaccia dell'istanza concreta `actualState`. Il pattern non definisce dove vada inserita la logica di cambiamento di stato, ma in generale pi flessibile consentire alle sottoclassi di `State` la definizione dello stato successivo.

L'applicazione del pattern consente di eliminare lunghe e complesse parti di codice in cui una sequenza di istruzioni condizionali determinano le istruzioni da eseguire in base allo stato dell'oggetto. Consente inoltre di isolare il codice specifico relativo ad uno stato in un singolo oggetto permettendo l'aggiunta di nuovi stati e transizioni semplicemente definendo nuove classi.

Bibliografia

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

[2] Kurose J. and Ross K. *Computer Networking - A Top-Down Approach.* Addison Wesley, 6th edition, 2012.