Indice

Elenco delle figure

Elenco delle tabelle

Acronimi

SF	Shadow Framework
GPU	Graphic Processing Unit
UML	Unified Modeling Language
fps	frame per secondo
API	Application Programming Interface
GPGPU	General-Purpose computing on Graphics Processing Units

Capitolo 1

Introduzione

Questo progetto di tesi è stato realizzato nell'ambito dello sviluppo dello Shadow Framework (SF) 2.0, un framework per lo sviluppo di applicazioni che fanno uso di grafica tridimensionale real-time ideato e sviluppato dall'Ingegner Alessandro Martinelli.

La grafica tridimensionale, o computer grafica 3d, consiste nell'utilizzo di modelli geometrici tridimensionali da parte di un computer per il calcolo e il rendering[1] di immagini digitali. Essa è attualmente molto diffusa ed utilizzata in moltissimi campi, cosa che la ha resa un'esperienza comune nella vita di tutti i giorni, non solo infatti ne viene fatto un uso intensivo nella pubblicità e nell'intrattenimento, ma anche in campi come la medicina e la ricerca scientifica.

La grafica tridimensionale real-time è un ramo specifico della grafica tridimensionale che si focalizza sulla generazione di simulazioni di ambienti e/o oggetti con la quale un utente può interagire osservando una reazione coerente nella simulazione. Questo senso di interazione viene fornito da una generazione sequenziale di immagini che, come nelle pellicole cinematografiche, danno un'illusione di movimento, ma in cui l'effettivo contenuto dei fotogrammi non è predeterminato ed è calcolato al momento sulla base degli input forniti.

In base a quanto esplicato in [2], per poter avere un'interazione soddisfacente è molto importante che la velocità con cui vengono visualizzate le immagini, misurata in frame per secondo (fps), si mantenga stabile ed il

[1] Con rendering, in computer grafica, si intende il processo che attraverso l'elaborazione di un modello determina il colore di ogni pixel contenuto in una immagine digitale.[7, 8]

più possibile elevata in modo da minimizzare i tempi di risposta ed impedire che questi interferiscano con l'interazione stessa. Ciò pone il problema di ottenere un alto valore di fps che a sua volta implica dei vincoli temporali sulla possibile elaborazione dei modelli tridimensionali, se infatti si volesse mantenere una velocità di visualizzazione pari a 60 fps, il tempo di calcolo a disposizione per l'elaborazione di un frame rispetto al precedente ammonta a circa 15 millisecondi. Questa caratteristica differenzia profondamente la grafica real-time da quella non-real-time, in cui non avendo vincoli temporali stretti per la generazione dei frame, il focus è invece spostato sull'applicazione di modelli di elaborazione complessi a volte anche computazionalmente onerosi, che siano in grado però di generare immagini il più possibile fotorealistiche.

Le problematiche della grafica real-time hanno imposto nel corso degli anni lo sviluppo di tecniche e tecnologie specifiche di settore il cui stato dell'arte è oggi rappresentato dall'ultima generazione di Application Programming Interface (API) di programmazione grafica, dalle Graphic Processing Unit (GPU)[2] a pipeline programmabile e dai Linguaggi di Shading.

Queste tecnologie hanno assunto una grande importanza perché non solo consentono di raggiungere elevatissimi livelli di qualità e prestazioni quando sono utilizzate per la generazione di grafica 3D real-time, ma la ricerca su di esse ha portato allo sviluppo del General-Purpose computing on Graphics Processing Units (GPGPU) ovvero la possibilità di utilizzare la capacità di calcolo delle GPU per processare anche dati differenti da quelli grafici.

Le GPU a pipeline programmabile, in contrapposizione a quelle con pipeline fissa, permettono di adattare gli stadi della pipeline di renderizzazione mediante l'utilizzo dei linguaggi di Shading. In sostanza l'hardware viene programmato per il calcolo di algoritmi specifici da applicare ai dati grafici. Ciò consente di adattare il processo di renderizzazione agli effetti che si desidera ottenere e di sfruttare l'hardware della GPU per velocizzarne la computazione. Questo paradigma si è rivelato molto efficace ed efficiente, tanto esso viene oggi applicato nella maggior parte delle GPU moderne, sia che si tratti di dispositivi di fascia alta che di processori grafici dedicati ad architetture mobile come i cellulari.

Parallelamente, le API di programmazione grafica moderne si sono svi-

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

luppate consentendo lo sfruttamento sempre più efficiente delle risorse hardware delle GPU, ma non solo: la loro integrazione su piattaforme pensate per il mercato embedded ha consentito di osservare una proliferazione di applicazioni che fanno uso di grafica tridimensionale sia su cellulari che tablet. Per questi dispositivi, dotati ormai quasi obbligatoriamente di fotocamera, non è difficile trovare applicazioni di realtà aumentata che sfruttino le API per inserire elementi grafici tridimensionali nelle immagini catturate dal sensore ottico. Un evento che nei prossimi anni avrà probabilmente un impatto molto significativo nel mercato consiste nel fatto che molti sviluppatori di browser per la navigazione di internet stanno attualmente lavorando per integrare l'API grafica WebGL tramite JavaScrip, all'interno dei loro software. Questo consentirà l'esecuzione di applicazioni 3D direttamente all'interno dei browser stessi, eliminando la transizione tra contenuti 3D e contenuti non-3D e consentendo una integrazione diretta con servizi internet di terze parti.

Di pari passo i produttori di middleware, engine e framework[3] per applicazioni di grafica hanno integrato nelle loro funzionalità la capacità di sfruttare le pipeline programmabili e tool per testare e comporre nuovi shader[4]. Una sempre maggiore quantità di queste case produttrici supportano le piattaforme mobile (come Android e iOS) e rilasciano plugin o applicazioni WebPlayer per distribuire contenuti attraverso i browser web. In alcuni casi, come quello citato in [6], produttori di browser e case di sviluppo collaborano per migliorare le prestazioni di WebGL e convertire in JavaScript gli engine e i framework per eseguili direttamente nelle pagine web.

In questo contesto si inserisce lo Shadow Framework. Esso è stato progettato non solo per utilizzare e supportare tutte le tecnologie che costituiscono lo stato dell'arte nel campo della grafica 3D real-time, ma il suo obbiettivo è quello di farlo sperimentando un nuovo approccio nella generazione e nella gestione dei dati grafici tridimensionali. La chiave di questo approccio consiste nell'abbandonare la vecchia concezione per cui gli oggetti tridimensionali sono scolpiti generando mesh di vertici che compongono triangoli, ma uti-

[3]Nel contesto delle applicazioni per la grafica tridimensionale con middleware di solito si intendono componenti software dedicate a compiti specifici, come la gestione della fisica o il pathfinding, che vengono affiancate agli engine ed ai framework. Per una descrizione di cosa si intende e le differenze tra engine e framework fare riferimento alla sezione 2.1

[4]Uno shader è un programma scritto con un linguaggio di shading, che viene caricato ed eseguito in hardware da una GPU.

lizzare primitive parametriche più complesse dei semplici triangoli per poi sfruttare le capacità delle pipeline programmabili e far generare dall'hardware stesso le mesh di punti necessarie per il processo di renderizzazione. Questo tipo di procedimento permette di adattare la qualità del risultato visivo finale in base alle prestazioni dell'hardware a disposizione senza la necessità di produrre diverse versioni dello stesso modello la cui unica differenze consiste nel numero di vertici. È infatti sufficiente programmare la pipeline per produrre un minor numero di vertici per alleggerire il calcolo, mantenendo inalterato il modello parametrico di partenza. Oltre hai vantaggi descritti, l'utilizzo di primitive parametriche permette di produrre file di dimensione molto contenuta rispetto ai sistemi classici in cui i file contengono l'elenco dei vertici che descrivono l'oggetto. Lo stesso paradigma viene usato all'interno del framework non solo per i modelli 3D, ma anche per la generazione delle texture da applicarvi, garantendo un meccanismo per scalarne la qualità in base alle esigenze.

Sebbene una descrizione più dettagliata delle caratteristiche del framework venga fornita nel capitolo 2 vale la pena citare alcune delle sue caratteristiche salienti come:

- la possibilità di definire nuove primitive grafiche con un linguaggio specifico del framework;
- la grande portabilità in quanto sviluppato in linguaggio Java;
- il design web-oriented, infatti i meccanismi interni sono stati progettati per supportare l'utilizzo di dati in remoto ed è in corso di realizzazione una versione del framework in JavaScript;

1.1 Obbiettivo del progetto

L'obbiettivo del progetto di tesi nasce dall'idea di produrre un'applicazione dimostrativa delle funzionalità di rete offerte dallo Shadow Framework. Ciò 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 dovuti a vincoli di sincronizzazione non evidenziati dai test effettuati con dati sulla macchina locale.

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

La progettazione di questi moduli si è focalizzata su alcuni punti cardine che rappresentano la chiave dell'aspetto di sviluppo legato al progetto. Dato che la struttura del framework è 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'estendibilità e il riutilizzo del codice.

1.2 Considerazioni generali

MENZIONARE I TEST

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

1.3 Organizzazione del documento

Il presente documento è 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 cui è 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 funzionalità offerte e i package java prodotti..
- **Capitolo 5:** in cui sono presentate le applicazioni di test ed i risultati prodotti.
- **Capitolo 6:** in cui viene presentato un riassunto del lavoro svolto, i risultati e vengono proposti alcuni sviluppi futuri.

Capitolo 2

Lo Shadow Framework 2.0

Questo capitolo presenta inizialmente una panoramica sugli attuali sistemi per la realizzazione di applicazioni che fanno uso di grafica tridimensionale real-time. Successivamente viene presentato lo ShadowFramework 2.0, le sue caratteristiche, la sua struttura e i suoi sviluppi futuri.

2.1 Titolo provvisorio

Lo scopo della presente tesi di laurea è quello di produrre dei moduli di estensione per lo Shadow Framework utili alla realizzazione di applicativi 3D orientati al web. Per lo stesso scopo esistono sul mercato una grande quantità di soluzioni le cui caratteristiche possono essere notevolmente differenti in base al target di sviluppo a cui si rivolgono.

Prima di fornire una descrizione di alcune di queste soluzioni è bene introdurre alcuni concetti utili a comprendere quali sono i punti focali che distinguono un prodotto da un altro. L'elemento fondamentale di ognuno di questi questi software è l'**engine**, altrimenti detto rendering engine o 3D-engine. Questo è il nucleo di ogni programma grafico e fornisce una serie di meccanismi che, sfruttando le API di programmazione grafica, permettono di disegnare su schermo i modelli tridimensionali voluti. Quando un prodotto consiste nel solo engine viene fornito nella forma di una libreria che permette permette di interagire con esso e configurarlo.

Dato che la più grande fetta di mercato per queste soluzioni software proviene dallo sviluppo di videogames sono nati una serie di software definiti **game engine**, questi sono veri e propri framework per la produzione di

applicazioni 3D e, oltre al rendering engine, integrano al loro interno moduli per la gestione della fisica, del suono, delle animazioni e di tutte quelle componenti utili per lo sviluppo di giochi per computer. Questi framework possono essere forniti sia sotto forma di libreria da integrare nel proprio codice che con tool grafici che permettono di creare la propria applicazione da zero, utilizzando solamente gli strumenti forniti.

Seguendo questa classificazione e dato che l'intento dello Shadow Framework è fornire un set di librerie completo per la creare applicazioni grafiche, con moduli per gestire animazioni e dati oltre che alla semplice renderizzazione, senza però vincolare all'utilizzo di librerie specifiche per elementi come l'intelligenza artificiale, il suono o altro che non sia direttamente coinvolto nella gestione della grafica, esso si trova a metà strada tra un rendering engine puro e un game engine.

Vengono ora presentati alcuni dei prodotti presenti sul mercato, la scelta di quali di essi presentare rispetto ad altri che non verranno citati non riflette la loro qualità, ma sono stati scelti in base rappresentativa.

2.1.1 Game engine per Adobe Flash

In questo caso invece che un prodotto specifico si preferisce citare una categoria di software. Per l'ambiente di Adobe esistono una grande quantità di game engine che sfruttano le API Flash per effettuare il rendering dei contenuti. Sebbene questo framework venga utilizzato da molti anni per la realizzazione di applicativi e che per lungo tempo abbia rappresentato l'ambiente di riferimento per applicativi web 3D, solo dalla release 11 comincia a sfruttare l'accelerazione hardware per la grafica tridimensionale.

Attualmente l'accelerazione consente di sfruttare le GPU a pipeline programmabile, ma non consente di usare le modalità legacy a pipeline fissa. Inoltre non vengono rese disponibili le API grafiche per la programmazione nativa come OpenGL o DirectX, ma una API proprietaria di nome Stage3D che fornisce un'astrazione di livello più alto e semplifica la gestione delle risorse. Questo sebbene consenta di semplificare e unificare tutte le piattaforme compatibili rende necessarie delle limitazioni per conservare la compatibilità, rendendo inaccessibili le funzionalità più avanzate degli hardware moderni (ad esempio è supportato lo Shader Model solo fino alla versione 2.0 sebbene oggi sia disponibile la versione 5.0). Questi dati sono reperiti da [1].

2.1.2 Unity 3D

Unity 3D è un game engine completo che dal 2005 ad oggi ha accresciuto molto la sua popolarità. Uno dei maggiori punti di forza di questo ambiente consiste nella presenza di una discreta quantità di tool grafici integrati tra loro, per il controllo del workflow di sviluppo. Sono inoltre integrati molti moduli per controllare aspetti quali le animazioni, la verifica delle performance ed il networking. Un qualità di primo piano è la compatibilità di un elevato numero di piattaforme, consentendo di pubblicare le applicazioni realizzate su piattaforme mobile (android e iOS), desktop (Windows, Mac e Linux), console e web (via browser). A proposito di quest'ultimo caso la fruizione dei contenuti via web è effettuata con due diversi metodi: o tramite un player flash o tramite l'API Google Native Client del browser Chrome che permette l'esecuzione di codice nativo all'interno del browser. In generale Unity 3D è un game engine moderno con pieno supporto alle ultime tecnologie grafiche rendendolo uno prodotto molto rappresentativo. Oltre alle caratteristiche citate buona parte del suo successo è probabilmente dovuto alla grande comunità di sviluppatori, anche indipendenti, ed alla numerosa quantità di asset grafici disponibili.

2.1.3 Unreal Engine e Unigine engine

Entrambi questi prodotti sono game engine completi piuttosto famosi per l'elevatissima qualità delle loro capacità di rendering. Entrambi vengono infatti spesso utilizzati come benchmark per testare le capacità grafiche hardware. Entrambi questi progetti sono maturi e moderni essendo dotati di tool grafici specifici e supportando le più recenti architetture grafiche. Entrambi supportano la maggior parte delle piattaforme desktop, mobile e consolle. Sebbene L'Unreal engine supporti anche flash e L'Unigine no, sono presentati insieme in quanto esempio di nuovo approccio alla distribuzione via web: entrambe le compagnie produttrici di questi software stanno infatti investendo nel portare i loro framework a supportare API WebGL. Gli annunci ufficiali possono essere trovati a questi riferimenti [6, 3].

2.2 Struttura dello Shadow Framework 2.0

Capitolo 3

Gestione dei dati nello Shadow Framework 2.0

La gestione dei dati è un compito molto importante all'interno del framework. Attraverso l'utilizzo di un layer di gestione dati astratto, ogni mudulo del framework può 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 funzionalità messe a disposizione ed i principali package e moduli coinvolti.

3.1 L'astrazione della gestione dati

All'interno di un'applicazione SF l'unità base di dati può essere identificata con quello che viene definito SFDataset e di cui si può trovare una descrizione più dettagliata al paragrafo 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 è un oggetto *Singleton* che realizza un *Bridge* tra l'astrazione di reperimento dati e la sua implementazione concreta[1]. Ogni componente può 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

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

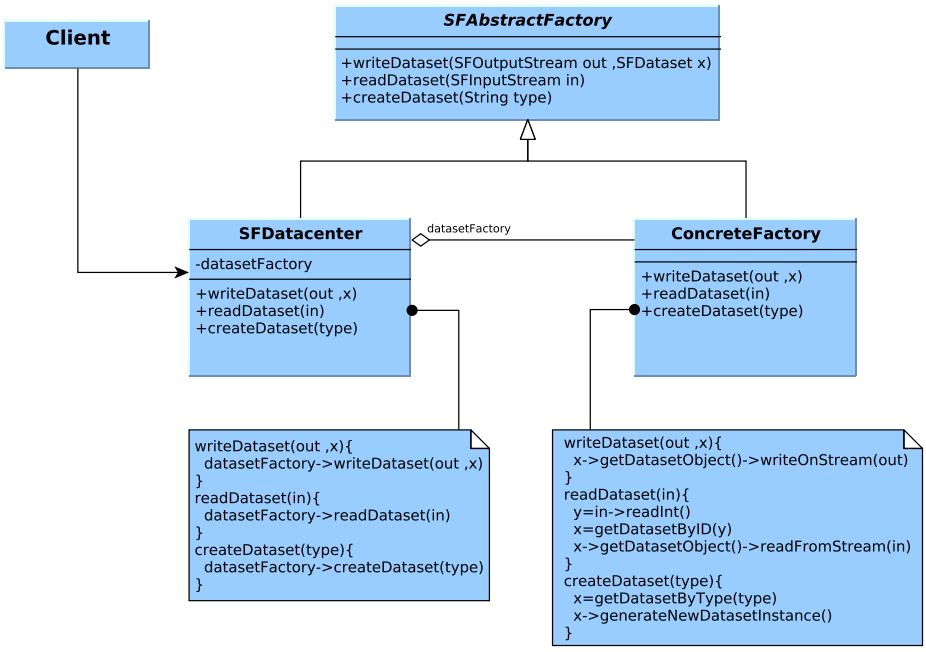

Figura 3.1: Diagramma del Bridge composto da SFDataCenter e da un'istanza concreta di SFAbstractFactory.

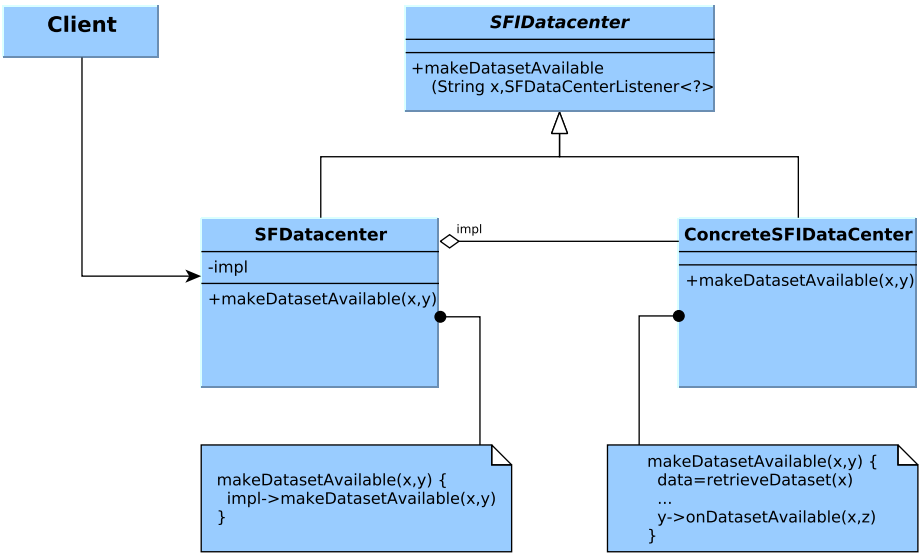

Figura 3.2: Diagramma del Bridge composto da SFDataCenter e da un'istanza concreta di SFIDataCenter.

di una nuova istanza di Dataset, identificata per tipo. L'oggetto Singleton espone queste funzionalità 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 esattamente come mostrato nelle immagini 3.1 e 3.2 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.

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

3.2.1 SFInputputStream e SFOutputStream

Queste interfacce definiscono le operazioni necessarie che uno stream di input o di output deve implementare affinché sia possibile leggere o scrivere su di esso dei DataObject e insieme costituiscono un elemento molto importante per l'estendibilità del framework sui dati. Su di esse si basa infatti l'astrazione che i dati utilizzano per completare le operazioni di lettura e scrittura. Utilizzando la medesima interfaccia di astrazione è possibile far comunicare tra loro anche implementazioni diverse del framework.

3.2.2 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*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 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.

3.2.3 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. Da come si può intuire dalla figura 3.3 il DataObject viene sfruttato dal Dataset per incapsulare i suoi dati interni

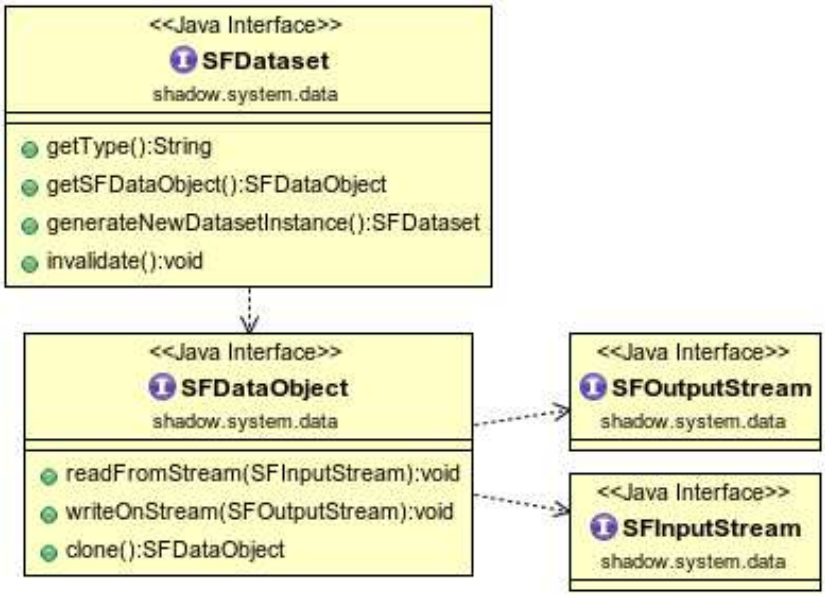

Figura 3.3: Diagramma della relazione tra le classi SFDataset, SFDataObject, SFInputStream e SFOutputStream.

ed incorporarne le funzionalità di lettura e scrittura. 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 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

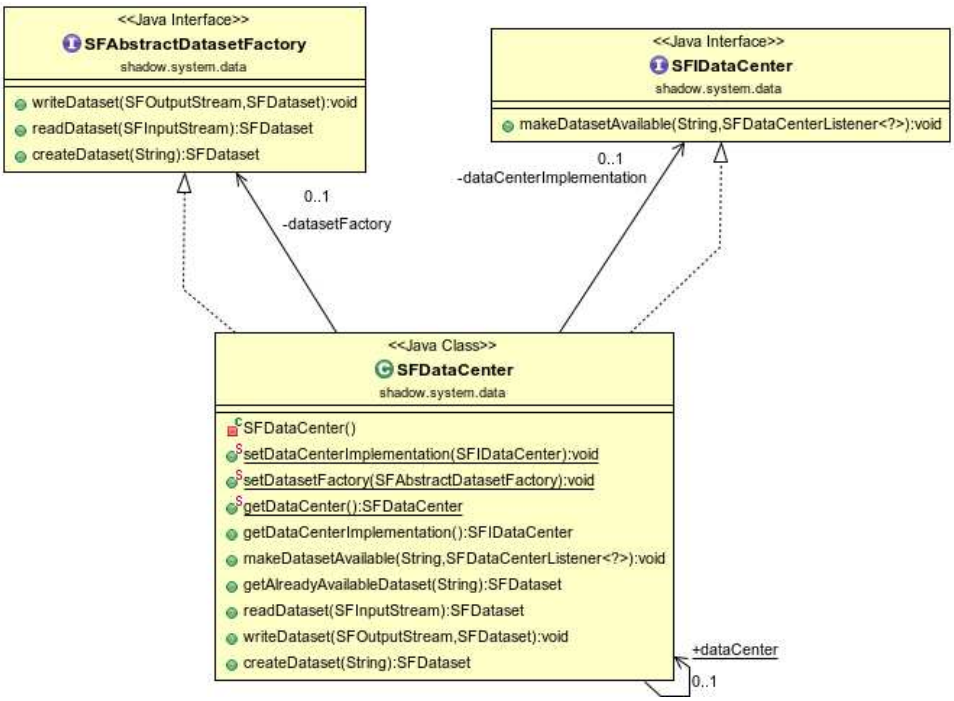

Figura 3.4: In questo diagramma viene mostrata la relazione tra le classi SFDataCenter, SFIDataCenter e SFAbstractDatasetFactory. Da notare che SFDataCenter è una classe Singleton in quanto contiene una istanza statica di se stessa (l'istanza dataCenter sottolineata) e possiede un costruttore privato (evidenziato in rosso). Inoltre, in riferimento alla sezione B.2.1, si può notare come SFDataCenter realizzi un Bridge sia con SFIDataCenter che con SFAbstractDatasetFactory

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 è 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 è pronto.

3.2.7 SFDataCenter

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

É 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`.

3.2.8 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.

3.3 Classi di utilità per il layer dati

3.3.1 SFLibraryreference

Un LibraryReference è un DataObject che può essere usato da qualsiasi componente per avere un riferimento ad un Dataset memorizzato in una libreria. Viene utilizzato all'interno di DataObject o di Dataset per non avere istanze doppie dello stesso dato.

3.3.2 SFGenericDatasetFactory

Questa classe di utilità consiste in una implementazione concreta di default dell'interfaccia `SFAbstractDatasetFactory`. Per consentire il riutilizzo del codice è 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.

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 è importante tener conto degli obbiettivi di progetto esposti al paragrafo 1.1 e della descrizione dei meccanismi di gestione dati interni del framework, descritti nel capitolo 3.

Nella sezione 4.1 del capitolo viene fornita innanzitutto una suddivisione e una descrizione dei moduli, nella sezione 4.2 viene descritto il meccanismo dei Dataset sostitutivi e nella 4.3 l'infrastruttura di ret e il protocollo di comunicazione. Infine è presente una panoramica sui package java e le principali classi di ognuno di essi nella sezione 4.4.

È giusto sottolineare che l'obbiettivo principale è stato la produzione di librerie e tool per lo sviluppo di applicazioni, focalizzandosi sull'estendibilità 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 è 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 è stata svolta in maniera distinta, ma si è trattato di un processo iterativo in cui i test hanno svolto più di una volta un ruolo di guida nel refactoring del codice.

Si rimanda all'appendice A per informazioni sul codice sorgente relativo al progetto e per informazioni sulle versioni delle librerie utilizzate.

4.1 Moduli

La libreria di classi realizzata può essere suddivisa in quattro macro-moduli suddivisi in base alle finalità e alle funzionalità. 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 utilità `GenericCommunicator` che oltre a consentire la gestione della connessione assegnatagli la utilizza per fornire funzionalità di lettura e scrittura di messaggi testuali attraverso il canale aperto.

Il modulo è 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 è `SFRemoteDataCenter`: questa è una classe di implementazione utilizzabile nel *Bridge* realizzato da `SFDataCenter`[1]. Dallo schema in figura 4.1 osserviamo che le richieste di Dataset effettuate

[1] Per la classe `SFDataCenter` si rimanda al paragrafo 3.2.7 mentre per il pattern *Bridge* al paragrafo B.2.1.

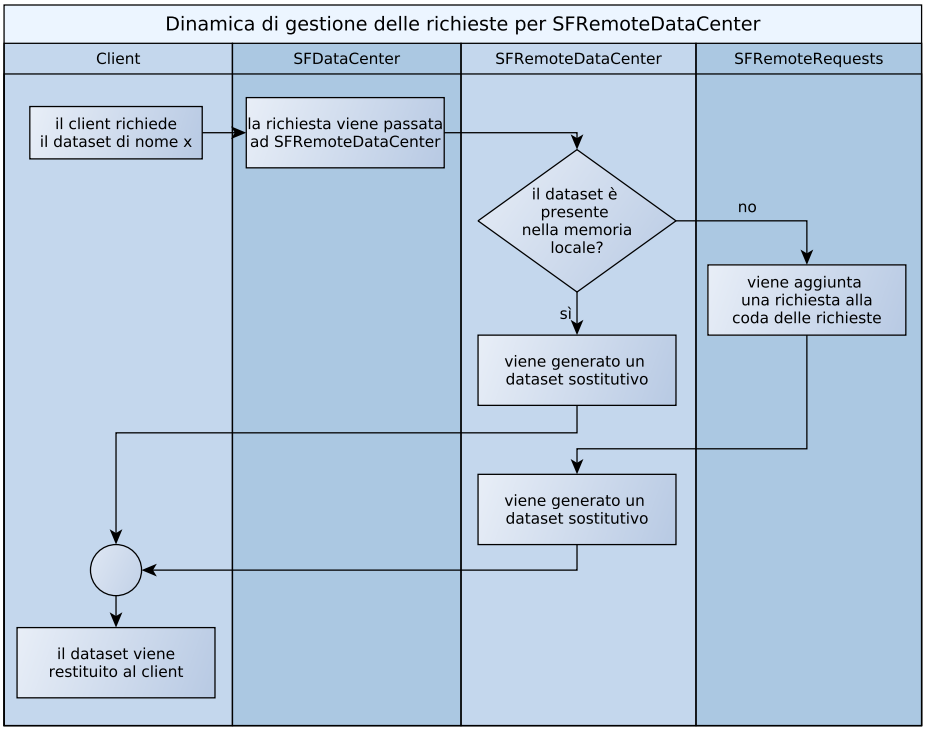

Figura 4.1: Lo schema mostra la sequenza di eventi dovuti ad una richiesta a SFDataCenter che utilizza un SFRemoteDataCenter come implementazione interna.

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 è presente viene generata una richiesta e aggiunta ad un buffer di richieste di nome SFRemoteRequests, successivamente al richiedente è restituito un Dataset sostitutivo temporaneo scelto opportunamente. Il buffer, che supporta la sincronizzazione, può contemporaneamente 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 è 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 delle 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 è pensata per il multi-threading. Il meccanismo messo a disposizione è mostrato nella figura 4.2, in esso un thread `RemoteDataCenterRequestsCreationTask` viene risvegliato periodicamente e controlla che non vi siano richieste pendenti nel buffer delle richieste. Se il buffer non è vuoto viene allora allocano un nuovo thread di tipo `RemoteDataCenterRequestTask` e mentre il precedente viene nuovamente sospeso questo effettua le richieste al server. Quando il secondo thread termina la comunicazione chiede al buffer di generare un update a tutti i client che avevano richiesto i dati reperiti e successivamente si chiude.

Il package che compone questo modulo è `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'im-

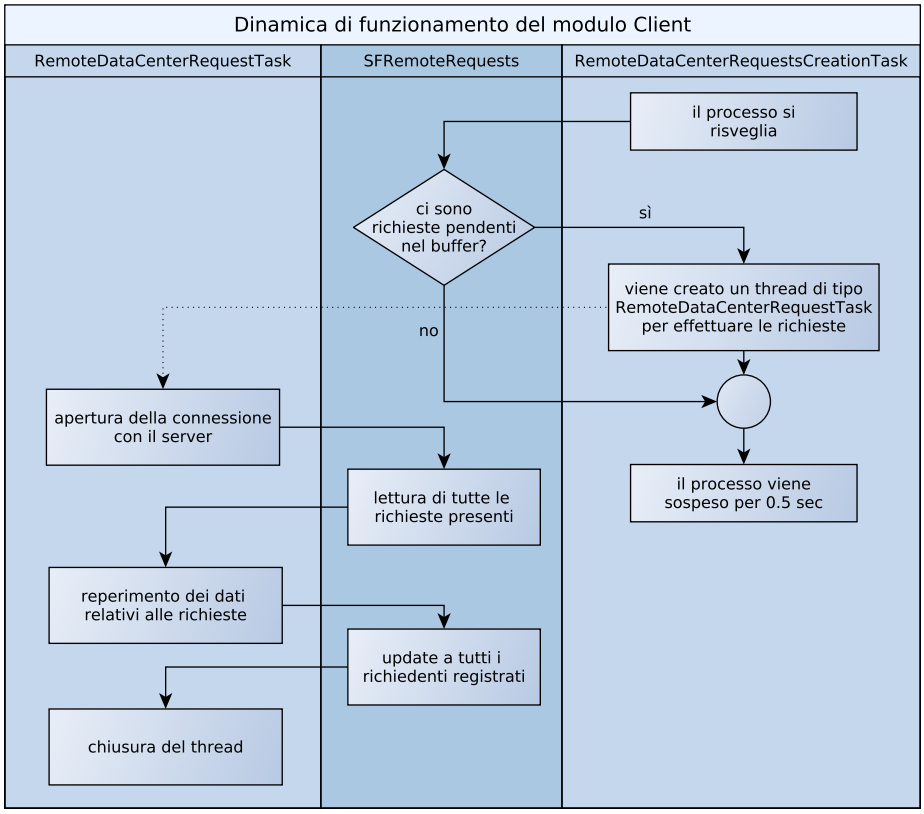

Figura 4.2: Dinamica del modulo client. Nello schema viene messa in evidenza la sostanziale indipendenza dei thread `RemoteDataCenterRequestsCreationTask` e `RemoteDataCenterRequestTask`, infatti a parte il fatto che i thread del secondo tipo sono creati dal primo non vi sono comunicazioni dirette tra i due.

plementazione è pensata per il funzionamento multi-thread in parallelo con l'applicazione principale che può così gestire più 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 è 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, è 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 più 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 è 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 è 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 è stata invece realizzata progressivamente durante l'implementazione dei test, quando si rendeva necessaria la costruzione di un Dataset specifico dato che quelli già 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 possibilità di estendere il protocollo di comunicazione in modo da rendere espandibili queste due librerie durante l'esecuzione, è 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 è 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 è ovvia la necessità di sviluppare uno strato dell'applicazione che si occupa della comunicazione di rete, da lato server si sono presentate 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 è stata anche la prima ad essere scartata, perché sebbene garantisse un tempo di messa in opera molto basso non consente un'estensione di funzionalità.

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

4.3.1 Protocollo di comunicazione

Una volta stabilito di utilizzare un server sviluppato appositamente è stato necessario stabilire le modalità 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 [5] mentre a livello inferiore viene utilizzato il protocollo TCP/IP. L'utilizzo del TCP in quanto protocollo confermato è 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'integrità dei dati sono prioritarie rispetto alle prestazioni.

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

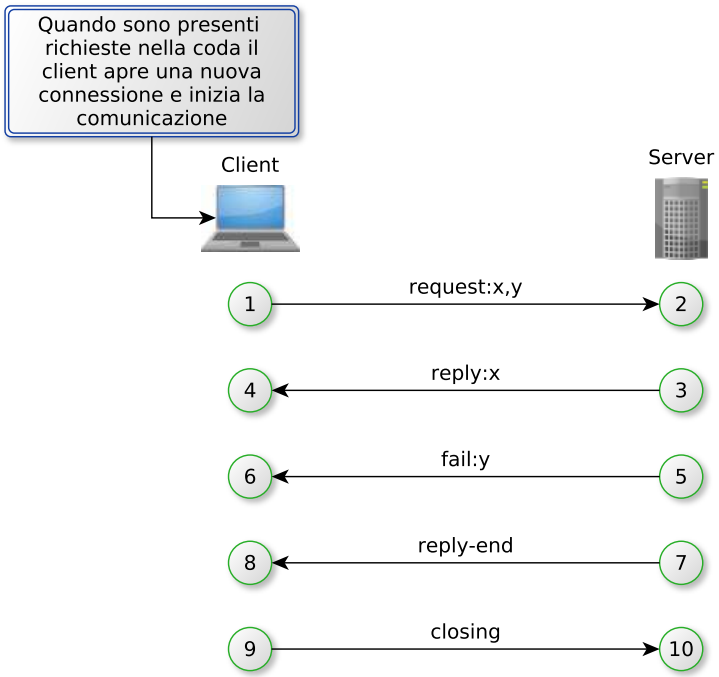

Figura 4.3: Esempio dello schema implementato per l'interazione tra client e server.

```
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 " : ".

Questi messaggi sono utilizzati all'interno di un preciso schema di interazione tra client e server di cui possiamo vedere un esempio nella figura 4.3, da notare che la numerazione dei nodi rappresenta la sequenza temporale dello scambio di messaggi. Lo schema presentato mostra che, quando la coda delle richieste non è vuota, il client apre una connessione con il server ed invia un messaggio di tipo **request** in cui sono elencati i nomi dei Dataset di cui ha bisogno. Il server a questo punto genera una sequenza di risposte attraverso cui invia, se possibile, i dati richiesti. Al termine della sequenza invia un messaggio di tipo **reply-end** a cui il client risponde con un messaggio **closing** per terminare la connnessione.

La gestione della comunicazione e dello schema di interazione viene ef-

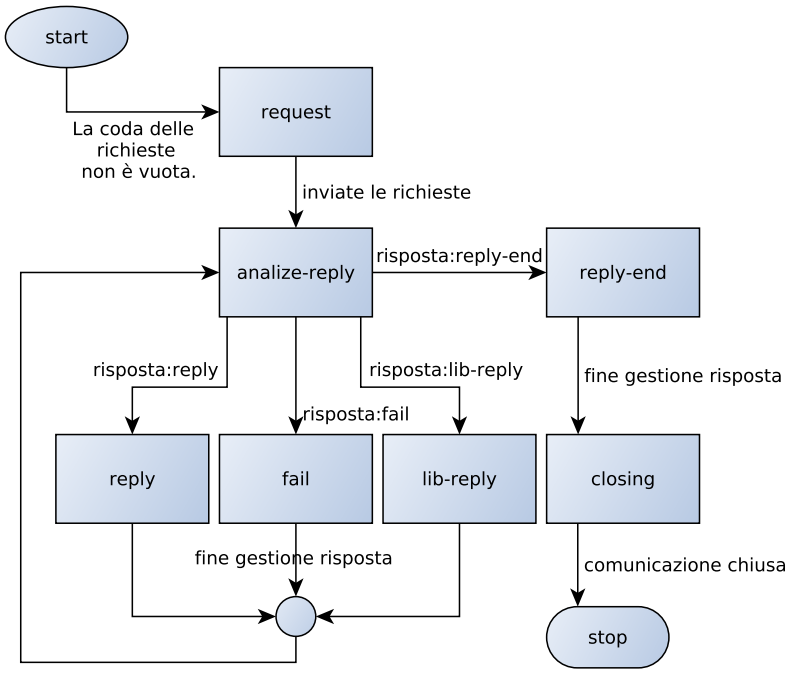

Figura 4.4: Diagramma a stati per la gestione della comunicazione delle richieste da parte del client.

fettuata 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 ciò ha però reso necessario strutturare l'implementazione delle macchine a stati in modo che fossero altrettanto espandibili. Per far questo è stato preso come modello il pattern *State*, descritto al paragrafo B.3.1, e la logica di ogni stato è stata incapsulata in un oggetto differente. Questo consente di espandere la macchina a stati semplicemente definendo nuovi oggetti i quali sono totalmente indipendenti dagli altri.

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 già presenti nel progetto SF20LiteTestWorld e usati per mostrare le capacità 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 è `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 più 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 ereditarietà 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 perché sfruttati dalle parti interessate del framework. Nella trattazione viene descritto l'obbiettivo di ogni schema, l'utilità, note implementative con schema Unified Modeling Language (UML) annesso e alcuni benefici della sua applicazione, vengono inoltre raggruppati per categoria così come vengono presentati in [4].

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 più piccole, rispetto ad oggetti gerarchici organizzati in base all'ereditarietà.

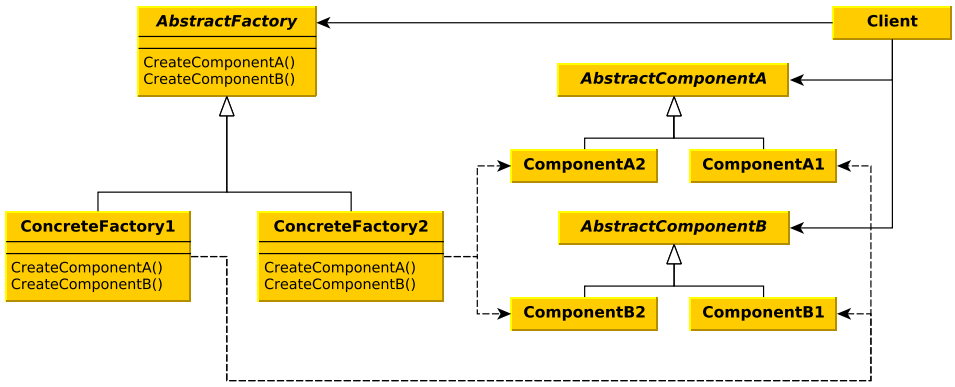

Figura B.1: Struttura UML del pattern Abstract Factory.

B.1.1 Abstract Factory

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

In questo modo un sistema può essere indipendente da come vengono creati gli oggetti concreti e può essere configurato per utilizzare diverse famiglie di oggetti a seconda delle necessità. Un caso esemplificativo è 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 però fondamentalmente identiche: un pulsante può 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 funzionalità.

Per ottenere ciò è 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 potrà utilizzare la Factory concreta che preferisce in modo del tutto indipendente da quale famiglia essa gestisce.

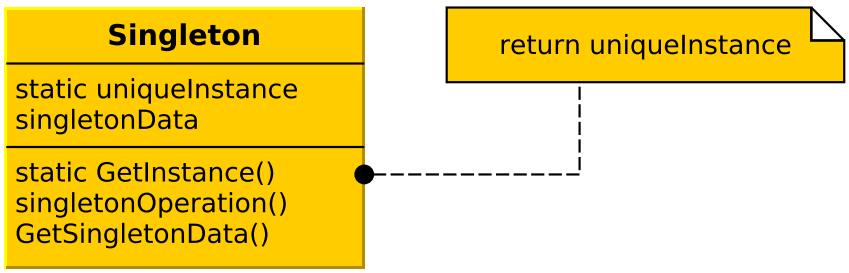

Figura B.2: Struttura UML del pattern Singleton.

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 è garantire che di una determinata classe possa essere creata una unica istanza, fornendone un singolo punto di accesso globale.

L'utilità di questo pattern è dovuta al fatto che spesso, all'interno di un'applicazione, ci sono servizi e funzionalità 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 proprietà spesso si delega alla classe stessa la responsabilità 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 in cui la classe possiede una unica istanza statica `uniqueIstance` accessibile solamente attraverso il metodo statico `GetInstance()`.

Utilizzare questo schema consente di avere un più stretto controllo degli accessi alla risorsa, inoltre estendendo la classe è possibile raffinare le funzionalità 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 più complesse. Un obbiettivo di questi pattern è permettere la composizione di oggetti complessi che uniscano le funzionalità di moduli più piccoli rendendo quest'ultimi il più 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.

Ciò è 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. É utile inoltre quando sia astrazione che implementazione hanno la necessità di essere estendibili attraverso sottoclassi.

Un utile esempio per illustrare questo pattern è prendere in esame un ipotetico toolkit per interfacce grafiche, per renderlo il più possibile portabile su piattaforme diverse esso deve utilizzare un'astrazione che descriva le finestre in modo più generico possibile. Se per ottenere queste proprietà usassimo semplicemente una classe astratta `Window` da cui, usando l'ereditarietà, 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 è la necessità di creare una finestra, deve essere istanziata una classe concreta specifica e questo lega fortemente l'astrazione con l'implementazione utilizzata rendendo più 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

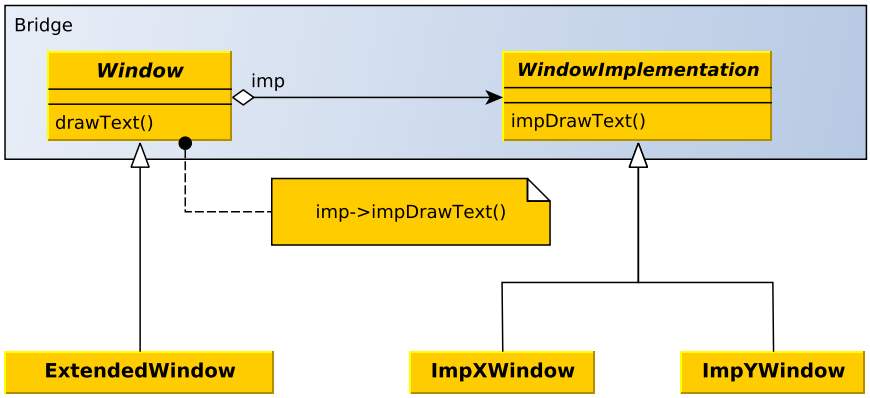

Figura B.3: Esempio di schema gerarchico dovuto all'applicazione del pattern Bridge.

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 verrà utilizzata dal client, la seconda è invece l'interfaccia che un'implementazione concreta deve utilizzare per essere utilizzata dall'astrazione. Come si può 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 così che il codice dei client non non ha un'effettiva necessità di essere ricompilato.

B.2.2 Composite

Lo scopo di questo pattern è 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 è un oggetto aggregato e ogni foglia è un oggetto semplice. Il nodo composito avrà al suo interno riferimenti ad oggetti figli che a loro volta possono essere semplici o compositi.

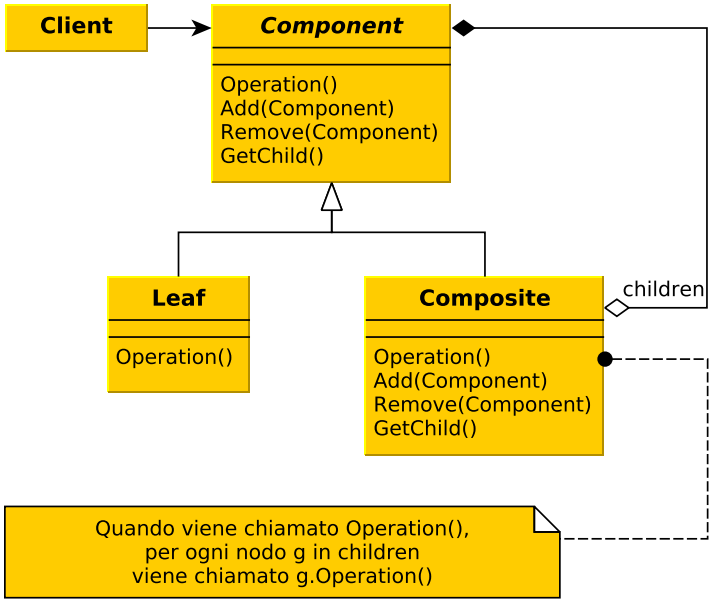

Figura B.4: Struttura UML del pattern Composite.

Spesso si desidera poter trattare oggetti composti nello stesso modo in cui gestiamo le sue componenti ignorando come siano effettivamente implementate le loro funzionalità, un esempio significativo sono le interfacce grafiche in cui è 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 è 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 può 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 risponderà direttamente in maniera appropriata mentre nel caso di un nodo verrà 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

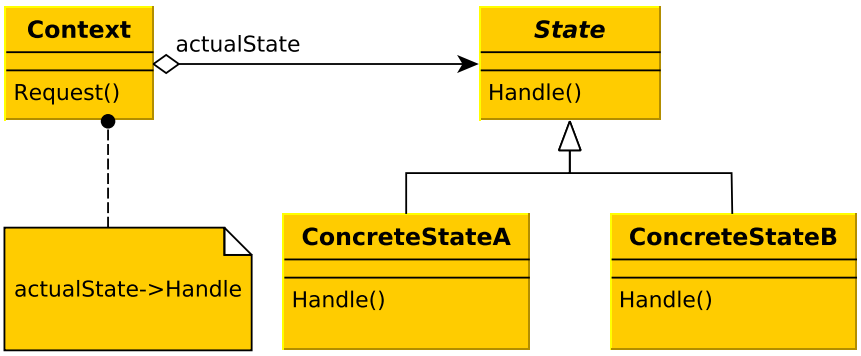

Figura B.5: Schema UML del pattern State.

codice pre-esistente.

B.3 Pattern Comportamentali

I pattern comportamentali riguardano gli algoritmi e la suddivisione delle responsabilità tra gli oggetti, per fare ciò 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 necessità 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 è più 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] Adobe. How stage3d works, 2013. [Online; controllata il 1-aprile-2013].

[2] Tomas Akenine-Mller, Eric Haines, and Naty Hoffman. *Real-Time Rendering*. A K Peters, Ltd., 3rd edition, 2008.

[3] UNIGINE Corp. Unigine's crypt demo is ported into a browser, 2013. [Online; controllata il 1-aprile-2013].

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

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

[6] Mozilla. Mozilla is unlocking the power of the web as a platform for gaming, 2013. [Online; controllata il 1-aprile-2013].

[7] Wikipedia. Rendering — wikipedia, l'enciclopedia libera, 2013. [Online; controllata il 1-aprile-2013].

[8] Wikipedia. Rendering — Wikipedia, the free encyclopedia, 2013. [Online; accessed 1-April-2013].