Indice

Elenco delle figure

Elenco delle tabelle

Acronimi

SF Shadow Framework

Parte I

Lo Shadow Framework

Capitolo 1

Gestione dei dati

Data management is an important task in the Framework. Through an abstract data layer, any module of the framework can be stored in datafile or transfered to datastream and loaded at any time.

1.1 Il package `shadow.system.data`

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

1.1.1 SFDataObject

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

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

I DataObject si basano sul *Composite Pattern*: possono essere semplici o contenere un insieme di oggetti figli, il fatto che sia gli oggetti complessi che gli oggetti semplici condividano la stessa interfaccia permette di trattare gli oggetti in modo uniforme. Un oggetto contenitore dovrà semplicemente richiamare lo stesso metodo di interfaccia per tutti gli oggetti figli i quali, se oggetti semplici, hanno la responsabilità di implementare l'algoritmo per leggere o scrivere se stessi da uno stream.

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

1.1.2 SFDataset

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

1.1.3 SFDataCenter

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

- `SFAbstractDatasetFactory`
- `SFIDataCenter`

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

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

1.1.4 SFObjectsLibrary

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

1.1.5 SFLibraryreference

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

Parte II

Progetto di Tesi

Capitolo 2

SFRemoteConnection

Appunti sparsi sul progetto

2.1 Obbiettivo del progetto

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

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

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

2.1.1 Infrastruttura di rete

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

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

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

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

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

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

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

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

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

Altri punti da considerare sono la difficoltà ed il tempo necessario per installare e configurare appropriatamente un server che potrebbe avere la ha al suo interno

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

2.2 SFRemoteConnection

2.3