SWEeneyThreads

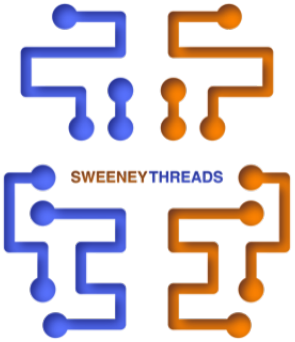
ActorBase

a NoSQL DB based on the Actor model

Specifica Tecnica

Redattori:
Bonato Paolo
Biggeri Mattia
Tommasin Davide

Approvazione:
Verifica:



Versione 1.0.3

6 aprile 2016

Indice

Diario delle modifiche

Versione	Data	Autore	Descrizione
1.0.3	2016-04-06	*Progettisti* Biggeri Mattia Tommasin Davide	Aggiunta sezione in appendice suoi Desing Pattern, contiene al momento la descrizione di: MVC, Event-driven, Command
1.0.2	2016-04-03	*Progettista* Bonato Paolo	Accorpate le sezioni "Componenti", "Package" e "Classi" in "Componenti e classi". Riadattata la sezione "Metodo e formalismo di specifica" alla nuova struttura. Inserite le immagini 1 e 2. Apportate le correzioni indicate.
1.0.1	2016-03-26	*Progettisti* Bonato Paolo Biggeri Mattia Padovan Tommaso Tommasin Davide Bortolazzo Matteo	Prima stesura di Architettura generale (sezinoe 3) e componenti (sezione 4)
1.0.0	2016-03-24	*Analisti* Bonato Paolo Biggeri Mattia	Creazione scheletro documento, stesura introduzione, definizione di metodo e formalismo di specifica.

Tabella 1: Diario delle modifiche

1 Introduzione

1.1 Scopo del documento

Il documento definisce la progettazione ad alto livello del progetto Actorbase. Verrà presentata l'architettura generale, le componenti, le classi e i design pattern utilizzati per realizzare il prodotto.

1.2 Scopo del prodotto

Il progetto consiste nella realizzazione di un DataBase NoSQL key-value basato sul modello ad Attori con l'obiettivo di fornire una tecnologia adatta allo sviluppo di moderne applicazioni che richiedono brevissimi tempi di risposta e che elaborano enormi quantità di dati. Lo sviluppo porterà al rilascio del software sotto licenza MIT.

1.3 Glossario

Al fine di evitare ambiguità di linguaggio e di massimizzare la comprensione dei documenti, il gruppo ha steso un documento interno che è il *Glossario v1.3.0*. In esso saranno definiti, in modo chiaro e conciso i termini che possono causare ambiguità o incomprensione del testo.

1.4 Riferimenti

- **Slide dell'insegnamento Ingegneria del software mod.A:**
  `http://www.math.unipd.it/~tullio/IS-1/2015/Dispense/E02.pdf`
- **Scala:**
  `http://www.scala-lang.org/`
- **Java:**
  `http://www.java.com/`
- **Akka:**
  `http://akka.io/`

1.4.1 Normativi

- **Norme di progetto:** *Norme di progetto v1.3.3*
- **Capitolato d'appalto Actorbase (C1):**
  `http://www.math.unipd.it/~tullio/IS-1/2015/Progetto/C1p.pdf`

2 Tecnologie utilizzate

2.1 Scala

Le possibili scelte dettate dal capitolato sono Java e Scala. Si è scelto di utilizzare Scala perché offre i seguenti vantaggi:

- **Concorrenza e distribuzione:** Ottimo supporto alla programmazione multi-threaded e distribuita, essenziale per la realizzazione di un prodotto responsive e scalabile.
- **Supporto di Akka:** Il linguaggio supporta la libreria Akka che è richiesta dal capitolato.

Inoltre il Committente ha espresso esplicitamente la sua preferenza sull'utilizzo di Scala.

2.2 Akka

L'utilizzo della libreria Akka è reso obbligatorio dal capitolato ed è la base del modello ad attori che costituisce il progetto.

3 Descrizione dell'architettura

3.1 Metodo e formalismo di specifica

Nell'esposizione dell'architettura del prodotto si procederà con un approccio di tipo top-down, ovvero dal generale al particolare. Inizialmente si descriveranno le tre componenti fondamentali: Client, Server e Driver; poi le componenti più piccole al loro interno, specificando i package e le classi che li compongono. Per ogni package saranno descritti brevemente il tipo, l'obiettivo e la funzione e saranno specificati eventuali figli, classi ed interazioni con altri package. Ogni classe sarà dotata di una breve descrizione e ne saranno specificate le responsabilità, le classi ereditate, le sottoclassi e le relazioni con altre classi. Successivamente saranno mostrati e descritti i diagrammi delle attività che coinvolgono l'utente. Infine si illustreranno degli esempi di utilizzo dei design pattern nell'architettura del sistema.

3.2 Architettura generale

Il sistema ha un'architettura generale di tipo client-server. Il server ha un'architettura di tipo event-driven basata sul modello ad attori ed espone delle API tramite socket TCP. L'architettura del Client segue il design pattern Model-View-Controller con interfaccia da linea di comando e comunica con il server grazie ad un driver tramite connessione TCP.

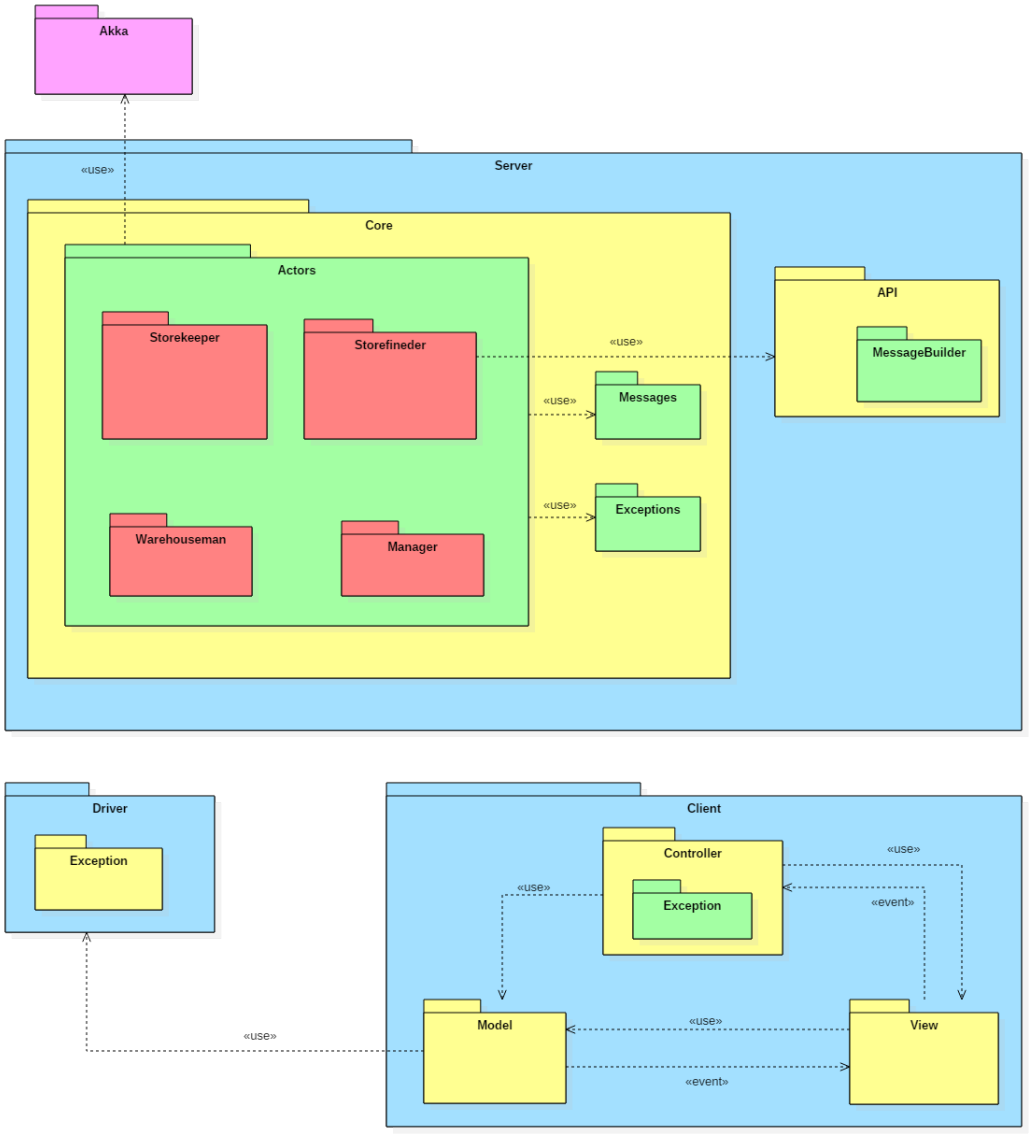


Figura 1: Architettura generale, vista Package

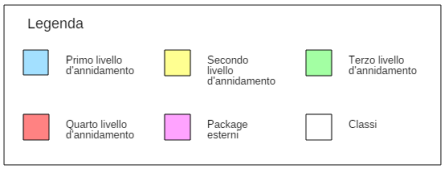


Figura 2: Legenda

3.2.1 Server

Le componenti principali del server sono gli attori:

- **Storekeeper:** Mantengono fisicamente in memoria le mappe chiave/valore.
- **Ninja:** Sono associati ad uno Storekeeper. Permettono di mantenere la consistenza del database anche nell'eventualità di un guasto. Se si perde uno Storekeeper, il Ninja a lui associato ne prende il posto. Per questa ragione, si trovano necessariamente in una macchina differente, ma rimangono costantemente in aggiornamento.
- **Storefinder:** Si occupano di ricevere richieste dall'esterno e instradarle ai rispettivi Storefinder, virtualmente ogni Storefinder definisce un indice sulla chiave della mappa. Il loro numero è veriabile e può essere configurato. Uno degli Storefinder è definito **main** e funge da punto di accesso al database.
- **Warehousemen:** Si interfaciano con gli Storekeeper e trascrivono persistentemente le rispettive mappe su disco.
- **Manager:** Ricevono le richieste di gestione degli attori di tipo Storekeeper, ad esempio sono responsabili del numero massimo di coppie chiave/valore contenute in un'istanza di Storekeeper.

3.2.2 Client

L'architettura del Client seguirà il design pattern MVC:

- **Model:** Il Model è la componente che si occupa di comunicare con il server usando i metodi del driver e di notificare la View quando avviene un cambiamento nel suo stato.
- **View:** La View è la componente che interagisce con l'utente mediante interfaccia a linea di comando. L'utente può usare il DSL per interrogare il Model. La View esegue delle *state query* sul model per avere le informazioni aggiornate.
- **Controller:** Il Controller è la componente che esegue il parsing dei comandi del DSL inseriti nella View e li notifica al Model.

3.2.3 Driver

Il Driver è una libreria, invocando i metodi della quale è possibile effettuare richieste TCP verso le API esposte dal Server.

4 Componenti e classi

4.1 componente 1

4.1.1 Package

4.1.2 Classi

5 Diagrammi delle attività

6 Design pattern

7 Stime di fattibilità e di bisogno di risorse

8 Tracciamento

8.1 Tracciamento componenti-requisiti

8.2 Tracciamento requisiti-componenti

9 Appendice

9.1 Descrizione Desing Pattern

Segue, per ogni Desing Pattern utilizzato, la descrizione dello scopo, motivazione e applicabilità.

9.1.1 Event-driven

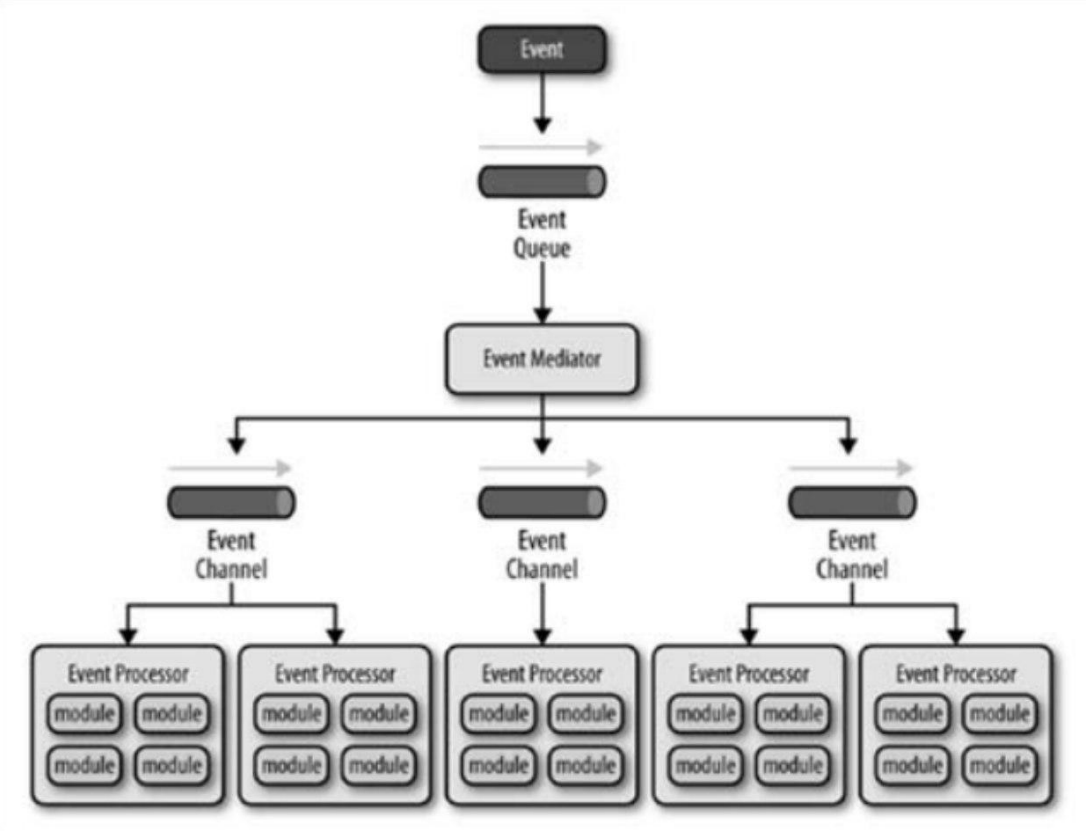


Figura 3: Diagramma del Desing Pattern Event-driven

- **Scopo:** Produrre applicazioni molto scalabili e processare eventi asincroni disaccoppiati.
- **Motivazione:** Gestire le richieste che vengono volte all' applicativo tramite eventi processati in modo asincrono.
- **Applicabilità:** Gestione di eventi attraverso l'utilizzo di un mediatore e elaboratori di eventi

9.1.2 MVC

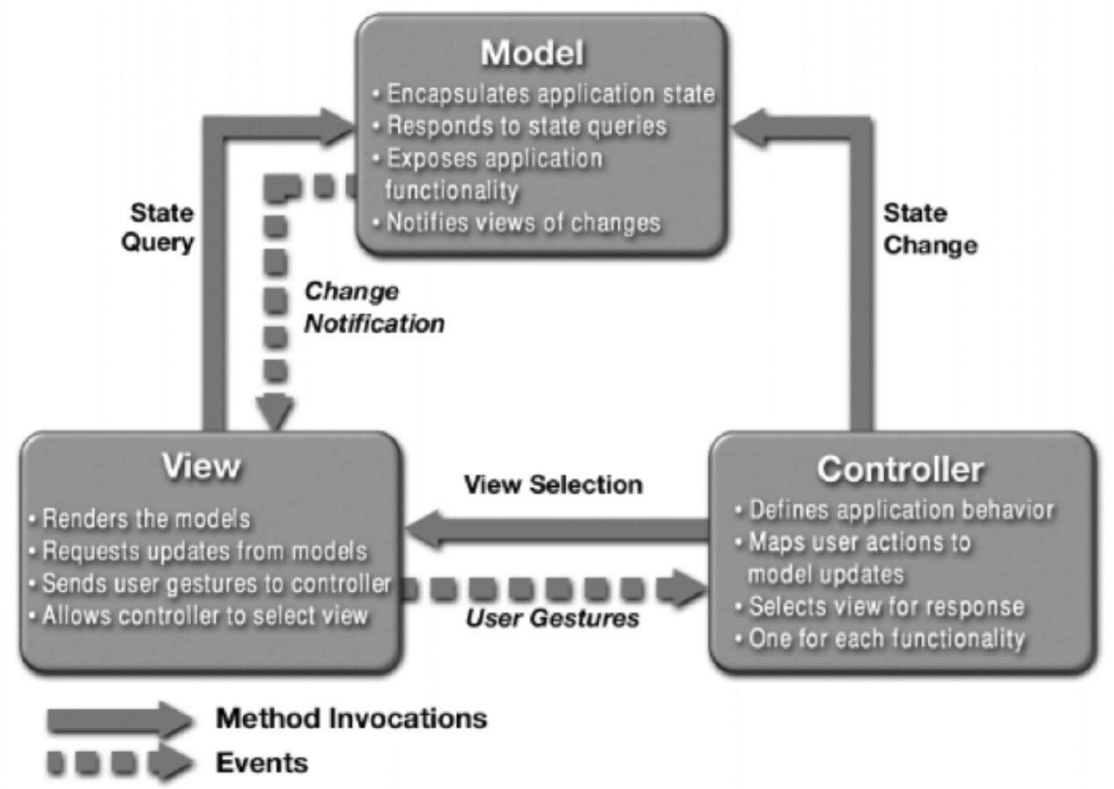


Figura 4: Diagramma del Desing Pattern MVC

- **Scopo:** Disaccoppiamento delle seguenti componenti:
  - Model regole di accesso e dati di business
  - View rappresentazione grafica
  - Controller reazioni della UI agli input utente
- **Motivazione:** Lo scopo di molti applicativi è di recuperare dati e mostrarli all'Utente. Si è visto che la migliore soluzione di questo scopo è dividere la modellazione del dominio, la presentazione e le reazioni basate sugli input degli utenti i tre classi separate, esistono vari desing pattern che svolgono questa separazione, uno di questi è MVC;
- **Applicabilità:**
  - Applicazioni che devono presentare attraverso una UI un insieme di informazioni
  - Le persone responsabili dello sviluppo hanno compentenze differenti

9.1.3 Command

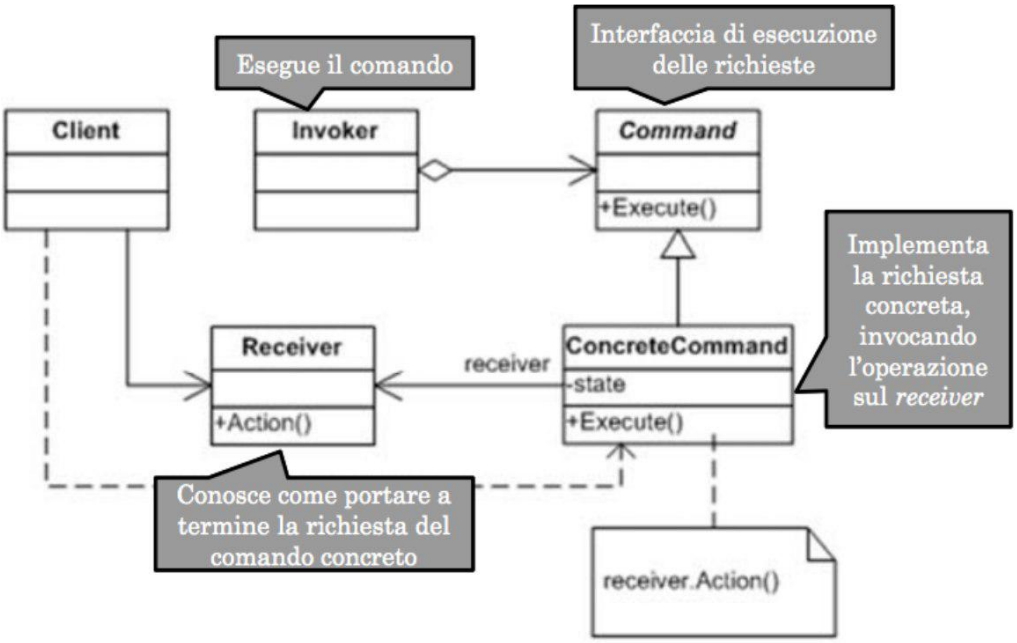


Figura 5: Diagramma del Desing Pattern Command

- **Scopo:**Incapsulare una richiesta in un oggetto, cosicché i client siano indipendenti dalle richieste
- **Motivazione:** Risolvere la necessità di gestire richieste di sui non si conoscono i particolari, tramite una classe astratta, Command, che definisce un interfaccia per eseguire la richiesta
- **Applicabilità:**
  - Parametrizzazione di oggetti sull'azione da eseguire
  - Specificare, accordare ed eseguire richieste molteplici volte
  - Supporto ad operazioni di Undo e Redo
  - Supporto a transazione, un comando equivale ad una operazione atomica

Elenco delle figure

Elenco delle tabelle