SWEeneyThreads

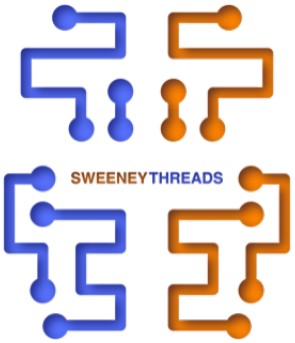
ActorBase

a NoSQL DB based on the Actor model

Specifica Tecnica

Redattori:
Bonato Paolo
Bortolazzo Matteo
Biggeri Mattia
Maino Elia
Nicoletti Luca
Padovan Tommaso
Tommasin Davide

Approvazione:
Verifica:



Versione 0.0.7

9 aprile 2016

Indice

1 Introduzione **6**
- 1.1 Scopo del documento . . . 6
- 1.2 Scopo del prodotto . . . 6
- 1.3 Glossario . . . 6
- 1.4 Riferimenti . . . 6
  - 1.4.1 Normativi . . . 6

2 Tecnologie utilizzate **7**
- 2.1 Scala . . . 7
- 2.2 Akka . . . 7

3 Descrizione dell'architettura **8**
- 3.1 Metodo e formalismo di specifica . . . 8
- 3.2 Architettura generale . . . 8
  - 3.2.1 Server . . . 9
  - 3.2.2 Client . . . 9
  - 3.2.3 Driver . . . 9

4 Componenti e Classi **11**
- 4.1 Actorbase . . . 11
  - 4.1.1 Descrizione . . . 11
  - 4.1.2 Package Figli . . . 11
- 4.2 Actorbase.Server . . . 12
  - 4.2.1 Descrizione . . . 12
  - 4.2.2 Package Figli . . . 12
  - 4.2.3 Classi . . . 12
- 4.3 Actorbase.Server.API . . . 12
  - 4.3.1 Descrizione . . . 12
  - 4.3.2 Classi . . . 13
- 4.4 Actorbase.Server.Core . . . 13
  - 4.4.1 Descrizione . . . 13
  - 4.4.2 Package figli . . . 13
- 4.5 Actorbase.Server.Core.Actors . . . 13
  - 4.5.1 Descrizione . . . 14
  - 4.5.2 Package figli . . . 14
- 4.6 Actorbase.Server.Core.Actors.DataManagement . . . 14
  - 4.6.1 Descrizione . . . 14
  - 4.6.2 Classi . . . 14
- 4.7 Actorbase.Server.Core.Actors.Manager . . . 15
  - 4.7.1 Descrizione . . . 15
  - 4.7.2 Classi . . . 15
- 4.8 Actorbase.Server.Core.Actors.StoreFinder . . . 15
  - 4.8.1 Descrizione . . . 15
  - 4.8.2 Classi . . . 15
  - 4.8.3 Interfacce . . . 16
- 4.9 Actorbase.Server.Core.Messages . . . 16
  - 4.9.1 Descrizione . . . 16
  - 4.9.2 Package Figli . . . 16
- 4.10 Actorbase.Server.Core.Messages.ConfigurationMessages . . . 17
  - 4.10.1 Descrizione . . . 17
  - 4.10.2 Classi . . . 17
  - 4.10.3 Interfacce . . . 17
- 4.11 Actorbase.Server.Core.Messages.PermissionMessages . . . 18
  - 4.11.1 Descrizione . . . 18
  - 4.11.2 Interfacce . . . 18
- 4.12 Actorbase.Server.Core.Messages.LinkActorsMessages . . . 18
  - 4.12.1 Descrizione . . . 18

4.12.2 Classi . . . 18
4.13 Actorbase.Server.Core.Messages.MainOperationMessages . . . 19
4.13.1 Descrizione . . . 19
4.13.2 Classi . . . 19
4.13.3 Interfacce . . . 19
4.14 Actorbase.Server.Core.Messages.DataManagerOperationMessages . . . 20
4.14.1 Descrizione . . . 20
4.14.2 Classi . . . 20
4.14.3 Interfacce . . . 20
4.15 Actorbase.Server.Core.Messages.ChangeInterfaceMessages . . . 20
4.15.1 Descrizione . . . 20
4.15.2 Classi . . . 21
4.16 Actorbase.Driver . . . 21
4.16.1 Descrizione . . . 21
4.16.2 Package Figli . . . 21
4.16.3 Classi . . . 21
4.16.4 Interfacce . . . 21
4.17 Actorbase.Driver.Connection . . . 22
4.17.1 Descrizione . . . 22
4.17.2 Utilizzo . . . 22
4.17.3 Relazione con altre classi . . . 22
4.18 Actorbase.Driver.Driver . . . 22
4.18.1 Descrizione . . . 22
4.18.2 Utilizzo . . . 22
4.18.3 Interfacce Estese . . . 22
4.18.4 Relazioni con altre classi . . . 22
4.19 Actorbase.Driver.Commands . . . 23
4.19.1 Descrizione . . . 23
4.19.2 Interfacce . . . 23
4.19.3 Classi . . . 24
4.20 Actorbase.Driver.Commands.ParsedCommand . . . 24
4.20.1 Descrizione . . . 24
4.20.2 Interfacce Figle . . . 24
4.21 Actorbase.Driver.Commands.ServerCommand . . . 24
4.21.1 Descrizione . . . 24
4.21.2 Interfacce Estese . . . 24
4.21.3 Interfacce Figlie . . . 24
4.22 Actorbase.Driver.Commands.ReadCommand . . . 25
4.22.1 Descrizione . . . 25
4.22.2 Interfacce Estese . . . 25
4.22.3 Classi Figlie . . . 25
4.23 Actorbase.Driver.Commands.WriteCommand . . . 25
4.23.1 Descrizione . . . 25
4.23.2 Interfacce Estese . . . 25
4.23.3 Classi Figlie . . . 25
4.24 Actorbase.Driver.Commands.ConnectionCommand . . . 26
4.24.1 Descrizione . . . 26
4.24.2 Interfacce Estese . . . 26
4.24.3 Classi Figlie . . . 26
4.25 Actorbase.Driver.Commands.DBSelectedCommand . . . 26
4.25.1 Descrizione . . . 26
4.25.2 Classi Figlie . . . 26
4.26 Actorbase.Driver.Commands.MapSelectedCommand . . . 26
4.26.1 Descrizione . . . 26
4.26.2 Interfacce Estese . . . 26
4.26.3 Classi Figlie . . . 27
4.27 Actorbase.Client . . . 27
4.27.1 Descrizione . . . 27

4.27.2 Package Figli . . . 27
4.28 Actorbase.Client.Model . . . 28
4.28.1 Descrizione . . . 28
4.28.2 Interfacce . . . 28
4.28.3 Classi . . . 28
4.29 Actorbase.Client.Model.Status . . . 29
4.29.1 Descrizione . . . 29
4.29.2 Utilizzo . . . 29
4.29.3 Classi Figlie . . . 29
4.30 Actorbase.Client.Model.ServerStatus . . . 29
4.30.1 Descrizione . . . 29
4.30.2 Utilizzo . . . 29
4.30.3 Interfacce Estese . . . 29
4.30.4 Relazioni con altre classi . . . 29
4.30.5 Classi Figlie . . . 29
4.30.6 Actorbase.Client.Model.ClientStatus . . . 29
4.30.7 Descrizione . . . 29
4.30.8 Utilizzo . . . 29
4.30.9 Interfacce Estese . . . 30
4.30.10 Relazioni con altre classi . . . 30
4.30.11 Classi Figlie . . . 30
4.31 Actorbase.Client.Model.Model . . . 30
4.31.1 Descrizione . . . 30
4.31.2 Utilizzo . . . 30
4.31.3 Relazioni con altre classi . . . 30
4.32 Actorbase.Client.Model.Comunication . . . 30
4.32.1 Descrizione . . . 30
4.32.2 Utilizzo . . . 30
4.32.3 Relazioni con altre classi . . . 30
4.33 Actorbase.Client.Model.ModelStatus . . . 30
4.33.1 Descrizione . . . 30
4.33.2 Relazioni con altre classi . . . 31
4.34 Actorbase.Client.Model.Help . . . 31
4.34.1 Descrizione . . . 31
4.34.2 Utilizzo . . . 31
4.34.3 Relazioni con altre classi . . . 31
4.35 Actorbase.Client.Model.ParsingFault . . . 31
4.35.1 Descrizione . . . 31
4.35.2 Utilizzo . . . 31
4.35.3 Relazioni con altre classi . . . 31
4.36 Actorbase.Client.View . . . 32
4.36.1 Descrizione . . . 32
4.36.2 Classi . . . 32
4.37 Actorbase.Client.View.View . . . 32
4.37.1 Descrizione . . . 32
4.37.2 Utilizzo . . . 32
4.37.3 Relazioni con altre classi . . . 32

5 Diagrammi delle attività **33**
5.0.1 Diagramma attività principale . . . 33
5.0.2 Offline Operation . . . 34
5.0.3 Connect to Server . . . 35
5.0.4 Online Operation . . . 36

6 Diagrammi di sequenza **37**
6.1 Avvio . . . 37
6.2 Chiusura . . . 38
6.3 Richiesta esterna . . . 38
6.4 Richiesta di creazione DB . . . 39
6.5 Richiesta di find . . . 40
6.6 Richiesta di aggiornamento item . . . 40
6.7 Creazione Ninja . . . 41
6.8 Creazione Ninja . . . 41
6.9 Sostituzione di uno Storekeeper . . . 42

7 Stime di fattibilità e di bisogno di risorse **43**

8 Tracciamento **44**
8.1 Tracciamento componenti-requisiti . . . 44
8.2 Tracciamento requisiti-componenti . . . 44

9 Appendice **45**
9.1 Descrizione Desing Pattern . . . 45
9.1.1 Event-driven . . . 45
9.1.2 MVC . . . 46
9.1.3 Command . . . 47
9.1.4 Singleton . . . 47
9.1.5 Singleton . . . 48

Elenco delle figure **49**

Elenco delle tabelle **50**

Diario delle modifiche

Versione	Data	Autore	Descrizione
0.0.5	2016-04-08	*Progettista* Nicoletti Luca	Inseriti tutti i diagrammi di sequenza riguardanti l'implementazione di richieste lato server, sostituita immagine dei Packages generale. Inserita una breve spiegazione di ogni diagramma di sequenza
0.0.6	2016-04-09	*Progettisti* Biggeri Mattia Padovan Tommaso Bonato Paolo	Stesura sezioni Driver, Diagrammi attività, Stime di fattibilità e bisogno di risorse; aggiunta immagine nella sezione 3.2.3, aggiunto Singleton nelle descrizioni dei desing pattern, aggiornata leggenda.
0.0.5	2016-04-08	*Progettisti* Maino Elia Nicoletti Luca Bortolazzo Matteo	Stesura sezione riguardante le componenti dell'architettura lato server: diagrammi dei package e delle classi e descrizioni testuali
0.0.4	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
0.0.3	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.
0.0.2	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)
0.0.1	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/`
- **InelliJ:**
  `http://www.jetbrains.com/idea/`

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:

- **Completamente Object-Oriented:** A differenza di Java, Scala è completamente orientato agli oggetti. Non c'è distinizione del tipo: oggetto - tipo primitivo, ogni valore è semplicemente un oggetto.
- **Staticamente tipato:** É un linguaggio tipato staticamente, questo permette di effettuare più facilmente i test. Inoltre Scala è in grado di stabilire il tipo di un oggetto per inferenza.
- **Può eseguire codice Java:** Scala può eseguire codice scritto in Java. É dunque possibile utilizzare classi e librerie scritte in Java all'interno di programmi scritti in Scala.
- **Concorrenza e distribuzione:** Ottimo supporto alla programmazione multi-threaded e distribuita, essenziale per la realizzazione di un prodotto responsive e scalabile.
- **Supporto alla definizione di DSL:** Scala supporta nativamente la definizione di DSL.
- **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.

Figura 1: Scala - logo

2.2 Akka

L'utilizzo della libreria Akka oltre ad essere reso obbligatorio dal committente, fornisce un'eccellente base su cui sviluppare un sistema basato sul modello ad attori. Akka permette di costruire facilmente applicazioni message-driven che siano estremamente concorrenti, distribuite e resilienti. La natura distribuita e asincrona degli attori messi a disposizione da Akka soddisfa pienamente i bisogni del sistema da implementare.

Figura 2: Akka - logo

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

L'architettura generale del sistema è 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.

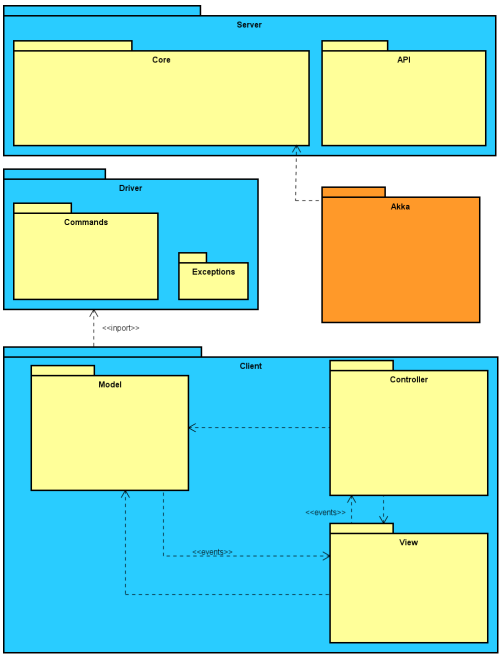


Figura 3: Architettura generale, vista Package

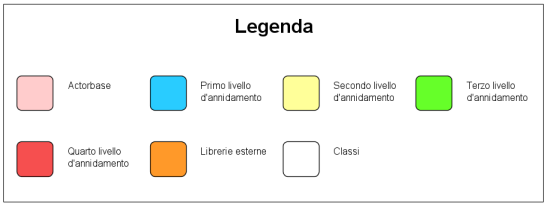


Figura 4: Legenda

3.2.1 Server

Il server di *Actorbase* è composto da due package principali: il package **Core** e il package **API**.
Il package **Core** è a sua volta composto dal package **Actors**, contenente le classi che definiscono gli attori del sistema, e dal package **messages**, contenente i messaggi che gli attori possono inviarsi tra loro.
Il package **API** contiene le classi che forniscono una comunicazione con i client esterni.

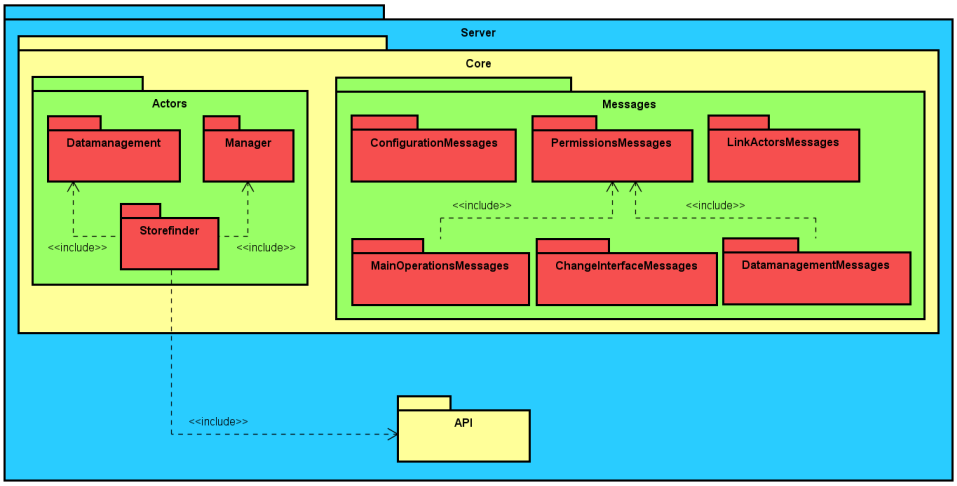


Figura 5: Server, vista Package

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.

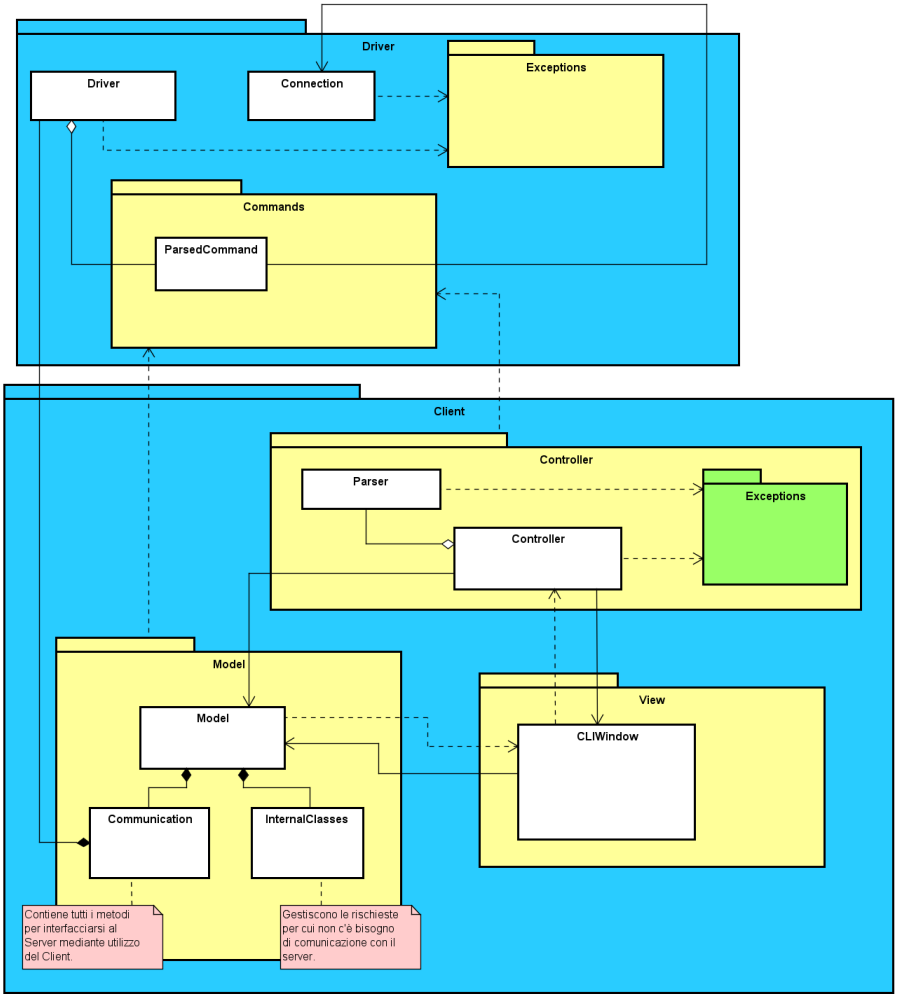


Figura 6: Architettura generale Client e Driver

4 Componenti e Classi

4.1 Actorbase

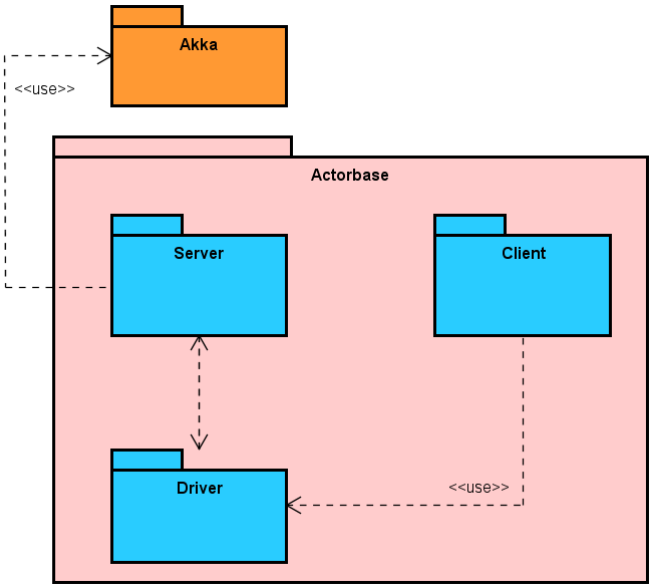


Figura 7: Componente Actorbase

4.1.1 Descrizione

É il package principale del sistema. L'interazione tra i package **Server** e **Driver** definiscono una comunicazione su rete di tipo client-server.
Le classi definite nel package **Server** utilizzano ed estendono le classi della libreria Akka.

4.1.2 Package Figli

- Actorbase.Server
- Actorbase.Client
- Actorbase.Driver
- Actorbase.Akka

4.2 Actorbase.Server

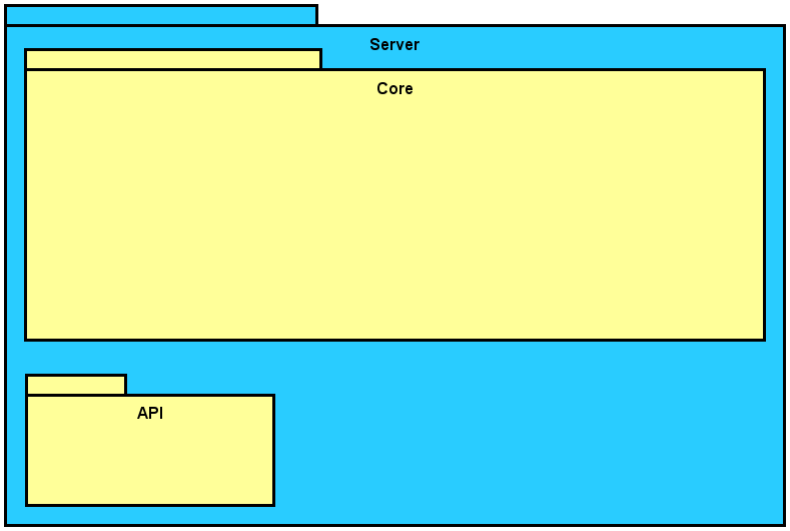


Figura 8: Componente Actorbase.Server

4.2.1 Descrizione

Package per la componente lato server del sistema. É composto dai packages **Core** ed **API** e dalla classe *ActorbaseServer*.

4.2.2 Package Figli

- Actorbase.Server.Core
- Actorbase.Server.API

4.2.3 Classi

- Actorbase.Server.ActorbaseServer

4.3 Actorbase.Server.API

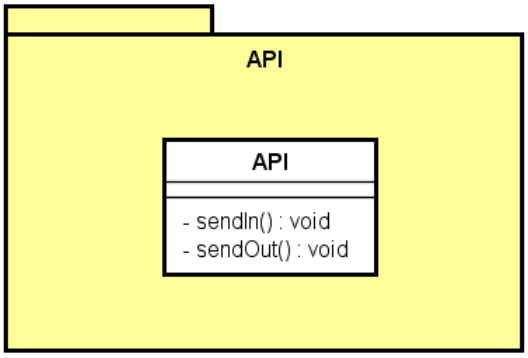


Figura 9: Componente Actorbase.Server.API

4.3.1 Descrizione

Package contenenti le classi che definiscono le API attraverso cui i client possono interfacciarsi all'istanza di un server del sistema.

4.3.2 Classi

- Actorbase.Server.API.API

4.4 Actorbase.Server.Core

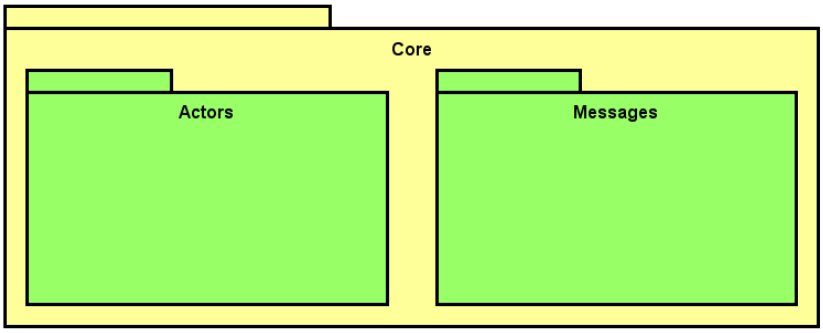


Figura 10: Componente Actorbase.Server.Core

4.4.1 Descrizione

Il package contiene le componenti che costituiscono il nucleo del sistema logico lato server. É composto da due package: **Actors** e **Messages**

4.4.2 Package figli

- Actorbase.Server.Core.Actors
- Actorbase.Server.Core.Messages

4.5 Actorbase.Server.Core.Actors

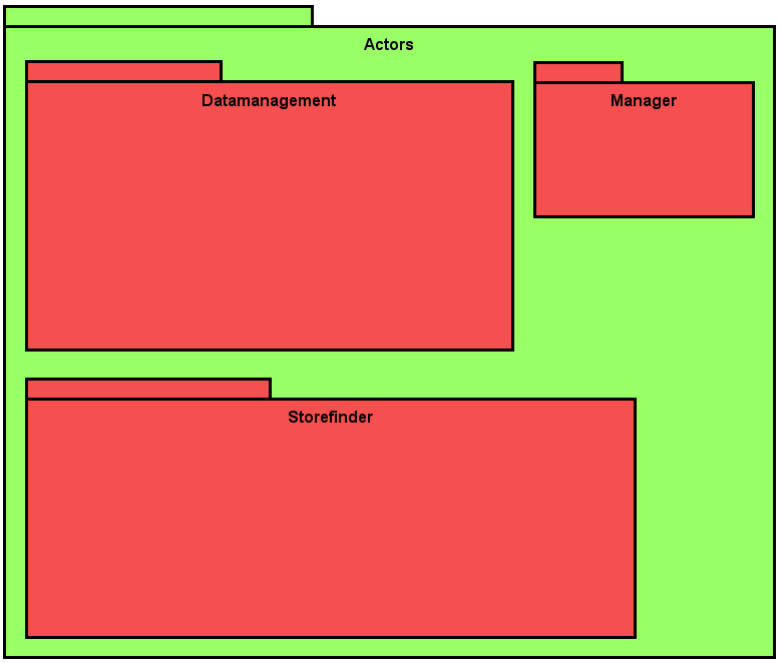


Figura 11: Componente Actorbase.Server.Core.Actors

4.5.1 Descrizione

Il package contiene le componenti che costituiscono i diversi attori definiti nel sistema. I package che lo compongono definiscono le diverse categorie degli attori.

4.5.2 Package figli

- Actorbase.Server.Core.Actors.DataManagement
- Actorbase.Server.Core.Actors.Manager
- Actorbase.Server.Core.Actors.StoreFinder

4.6 Actorbase.Server.Core.Actors.DataManagement

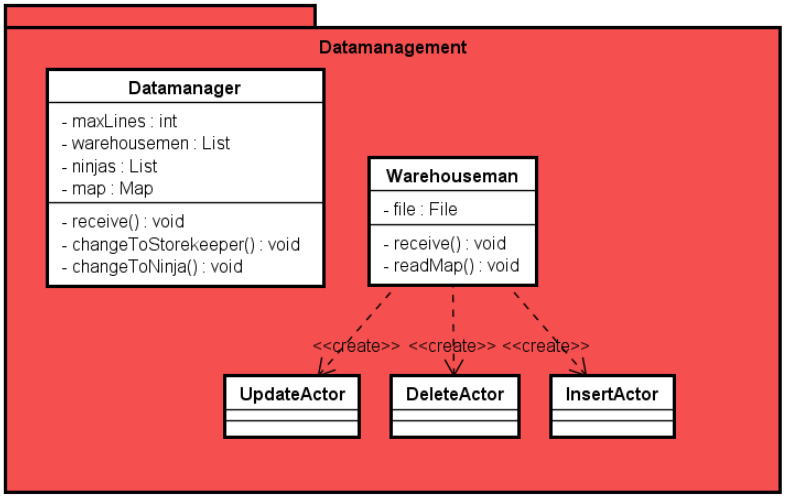


Figura 12: Componente Actorbase.Server.Core.Actors.DataManagement

4.6.1 Descrizione

All'interno di questo package sono definite le classi che rappresentano gli attori che si occupano direttamente della gestione dei dati.

4.6.2 Classi

- Actorbase.Server.Core.Actors.DataManagement.DataManager
- Actorbase.Server.Core.Actors.DataManagement.WareHouseMan
- Actorbase.Server.Core.Actors.DataManagement.UpdateActor
- Actorbase.Server.Core.Actors.DataManagement.DeleteActor
- Actorbase.Server.Core.Actors.DataManagement.InsertActor

4.7 Actorbase.Server.Core.Actors.Manager

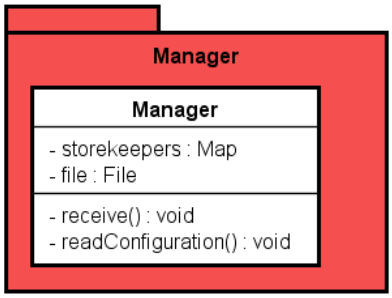


Figura 13: Componente Actorbase.Server.Core.Actors.Manager

4.7.1 Descrizione

All'interno di questo package sono definite le classi che rappresentano gli attori che si occupano della gestione di altri attori e dei vincoli presenti su di essi.

4.7.2 Classi

- Actorbase.Server.Core.Actors.Manager.Manager

4.8 Actorbase.Server.Core.Actors.StoreFinder

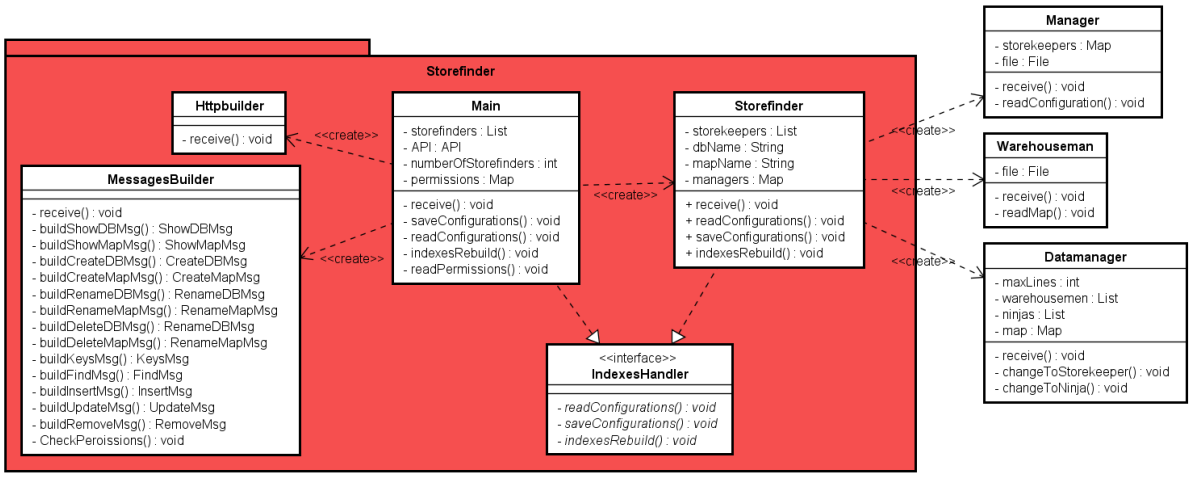


Figura 14: Componente Actorbase.Server.Core.Actors.StoreFinder

4.8.1 Descrizione

All'interno di questo package sono definite le classi che rappresentano gli attori che si occupano dell'indicizzazione degli altri attori presenti e del corretto instradamento dei messaggi.

4.8.2 Classi

- Actorbase.Server.Core.Actors.StoreFinder.StoreFinder
- Actorbase.Server.Core.Actors.StoreFinder.Main
- Actorbase.Server.Core.Actors.StoreFinder.HTTPBuilder
- Actorbase.Server.Core.Actors.StoreFinder.MessageBuilder

4.8.3 Interfacce

- Actorbase.Server.Core.Actors.StoreFinder.IndexesHandler

4.9 Actorbase.Server.Core.Messages

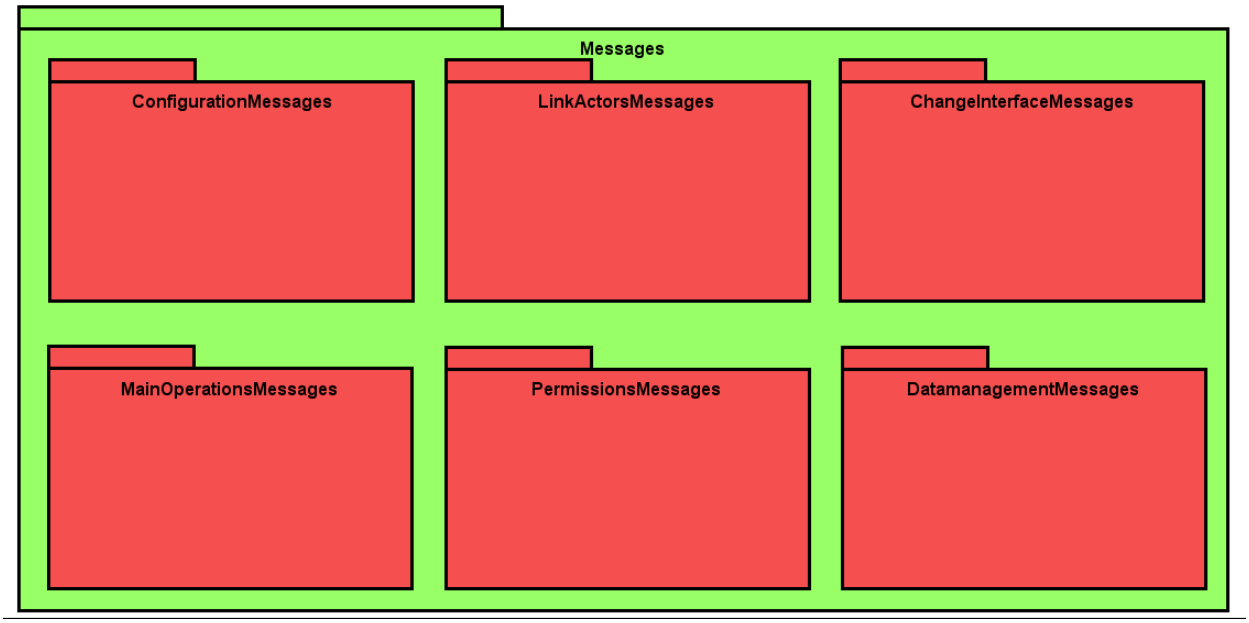


Figura 15: Componente Actorbase.Server.Core.Messages

4.9.1 Descrizione

All'interno di questo package sono definite le componenti che rappresentano i messaggi che i diversi attori del sistema possono inviarsi tra loro.

4.9.2 Package Figli

- Actorbase.Server.Core.Messages.ConfigurationMessages
- Actorbase.Server.Core.Messages.PermissionMessages
- Actorbase.Server.Core.Messages.LinkActorsMessages
- Actorbase.Server.Core.Messages.MainOperationMessages
- Actorbase.Server.Core.Messages.DataManagementMessages
- Actorbase.Server.Core.Messages.ChangeInterfaceMessages

4.10 Actorbase.Server.Core.Messages.ConfigurationMessages

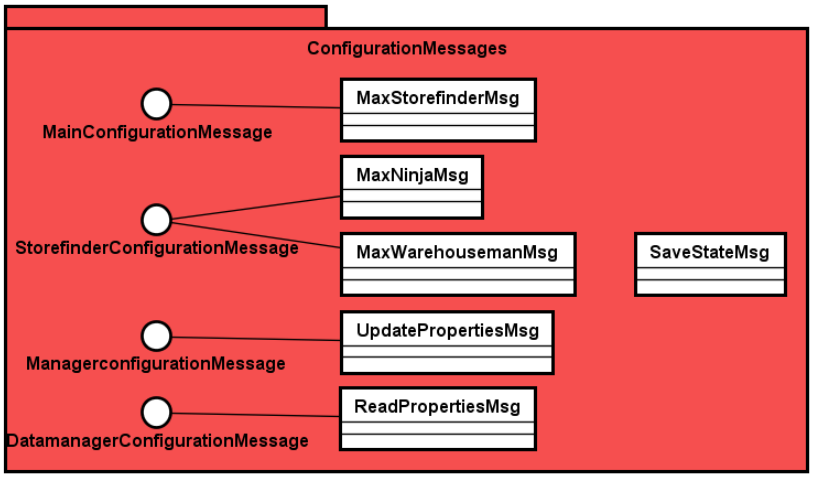


Figura 16: Componente Actorbase.Server.Core.Messages.ConfigurationMessages

4.10.1 Descrizione

All'interno di questo package sono definite le classi e le interfacce che rappresentano i messaggi relativi ad operazioni di configurazione delle impostazioni del server.

4.10.2 Classi

- Actorbase.Server.Core.Messages.ConfigurationMessages.MaxStoreFinderMsg
- Actorbase.Server.Core.Messages.ConfigurationMessages.MaxNinjaMsg
- Actorbase.Server.Core.Messages.ConfigurationMessages.MaxWarehousemanMsg
- Actorbase.Server.Core.Messages.ConfigurationMessages.UpdatePropertiesMsg
- Actorbase.Server.Core.Messages.ConfigurationMessages.ReadPropertiesMsg
- Actorbase.Server.Core.Messages.ConfigurationMessages.SaveStateMsg

4.10.3 Interfacce

- Actorbase.Server.Core.Messages.ConfigurationMessages.MainConfigurationMessage
- Actorbase.Server.Core.Messages.ConfigurationMessages.StoreFinderConfigurationMessage
- Actorbase.Server.Core.Messages.ConfigurationMessages.ManagerConfigurationMessage
- Actorbase.Server.Core.Messages.ConfigurationMessages.DataManagerConfigurationMessage

4.11 Actorbase.Server.Core.Messages.PermissionMessages

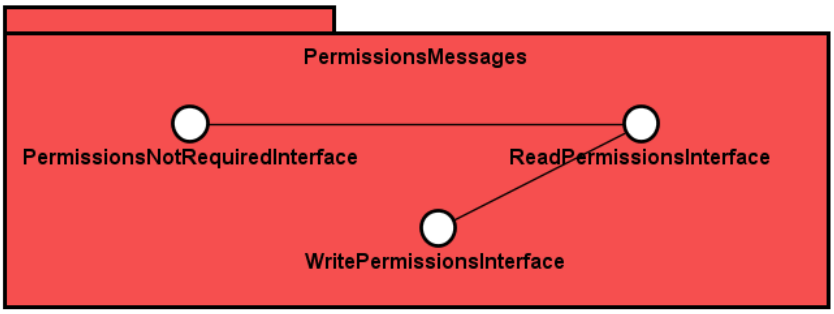


Figura 17: Componente Actorbase.Server.Core.Messages.PermissionMessages

4.11.1 Descrizione

All'interno di questo package sono definite le interfacce che rappresentano i diversi gradi di permesso che un'operazione richiede. Un'operazione può infatti richiedere i permessi di lettura, scrittura o nessun permesso. Ogni messaggio relativo ad un'operazione richiedibile da un client estende una di queste interfacce.

4.11.2 Interfacce

- Actorbase.Server.Core.Messages.PermissionMessages.PermissionsNotRequiredInterface
- Actorbase.Server.Core.Messages.PermissionMessages.ReadPermissionsInterface
- Actorbase.Server.Core.Messages.PermissionMessages.WritePermissionsInterface

4.12 Actorbase.Server.Core.Messages.LinkActorsMessages

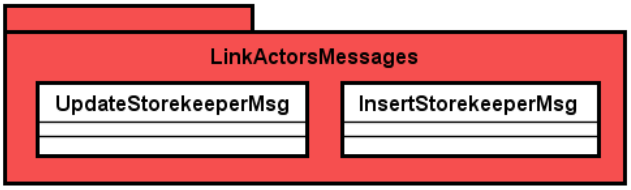


Figura 18: Componente Actorbase.Server.Core.Messages.LinkActorsMessages

4.12.1 Descrizione

All'interno di questo package sono definite le classi che rappresentano i messaggi relativi alla gestione dei collegamenti tra diversi attori.

4.12.2 Classi

- Actorbase.Server.Core.Messages.LinkActorsMessages.UpdateStoreKeeperMsg
- Actorbase.Server.Core.Messages.LinkActorsMessages.InsertStoreKeeperMsg

4.13 Actorbase.Server.Core.Messages.MainOperationMessages

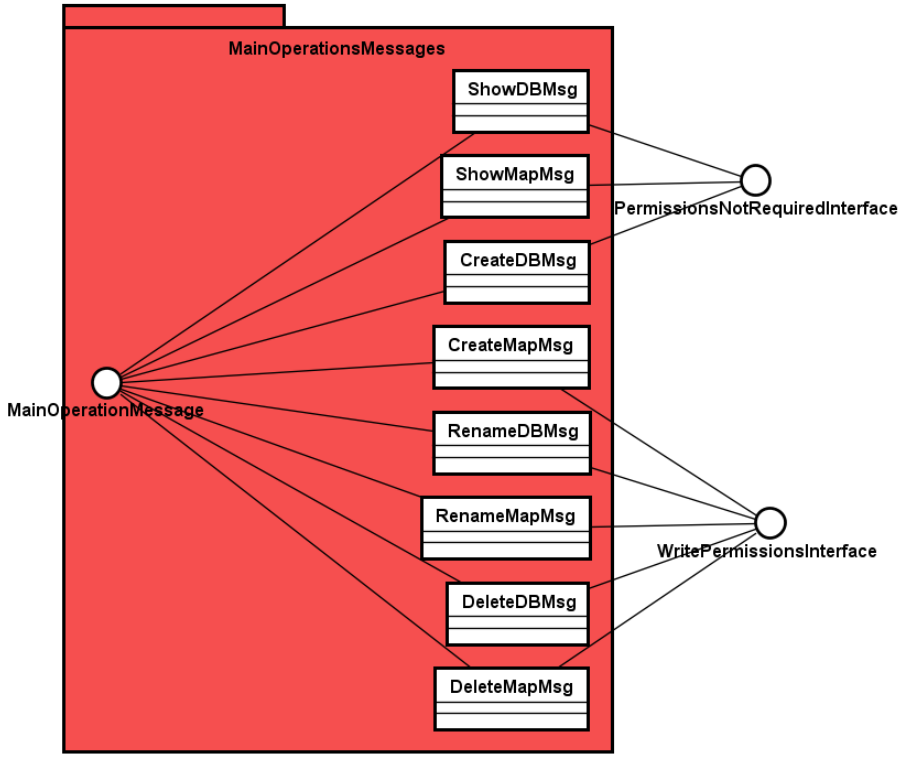


Figura 19: Componente Actorbase.Server.Core.Messages.MainOperationMessages

4.13.1 Descrizione

All'interno di questo package sono definite le classi e le interfacce che rappresentano i messaggi relativi ad operazioni che non richiedono l'invio di ulteriori messaggi ad attori che gestiscono i dati direttamente.

4.13.2 Classi

- Actorbase.Server.Core.Messages.MainOperationMessages.ShowDBMsg
- Actorbase.Server.Core.Messages.MainOperationMessages.ShowMapMsg
- Actorbase.Server.Core.Messages.MainOperationMessages.CreateDBMsg
- Actorbase.Server.Core.Messages.MainOperationMessages.CreateMapMsg
- Actorbase.Server.Core.Messages.MainOperationMessages.RenameDBMsg
- Actorbase.Server.Core.Messages.MainOperationMessages.RenameMapMsg
- Actorbase.Server.Core.Messages.MainOperationMessages.DeleteDBMsg
- Actorbase.Server.Core.Messages.MainOperationMessages.DeleteMapMsg

4.13.3 Interfacce

- Actorbase.Server.Core.Messages.MainOperationMessages.MainOperationMessage

4.14 Actorbase.Server.Core.Messages.DataManagerOperationMessages

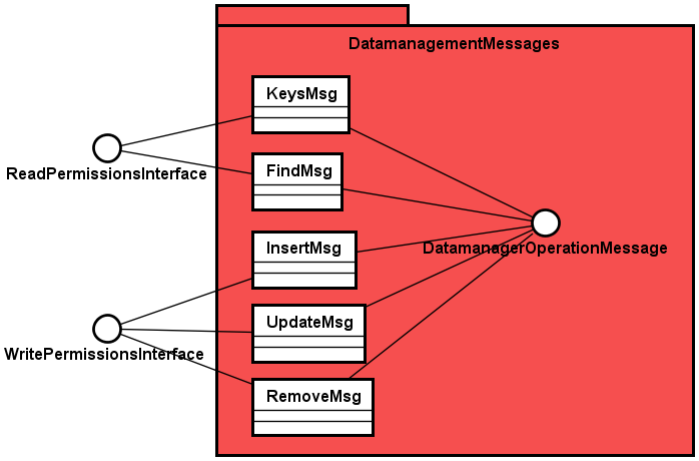


Figura 20: Componente Actorbase.Server.Core.Messages.DataManagerOperationMessages

4.14.1 Descrizione

All'interno di questo package sono definite le classi e le interfacce che rappresentano i messaggi relativi ad operazioni che richiedono l'invio di tali messaggi anche ad attori che gestiscono i dati direttamente.

4.14.2 Classi

- Actorbase.Server.Core.Messages.DataManagerOperationMessages.KeysMsg
- Actorbase.Server.Core.Messages.DataManagerOperationMessages.FindMsg
- Actorbase.Server.Core.Messages.DataManagerOperationMessages.InsertMsg
- Actorbase.Server.Core.Messages.DataManagerOperationMessages.UpdateMsg
- Actorbase.Server.Core.Messages.DataManagerOperationMessages.RemoveMsg

4.14.3 Interfacce

- Actorbase.Server.Core.Messages.DataManagerOperationMessages.DataManagerOperationMessage

4.15 Actorbase.Server.Core.Messages.ChangeInterfaceMessages

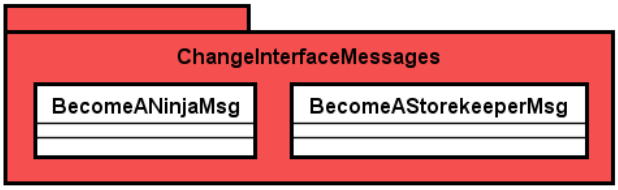


Figura 21: Componente Actorbase.Server.Core.Messages.ChangeInterfaceMessages

4.15.1 Descrizione

All'interno di questo package sono definite le classi e le interfacce che rappresentano i messaggi inviabili per effettuare operazioni di cambio interfaccia per gli attori che supportano tale funzionalità.

4.15.2 Classi

- Actorbase.Server.Core.Messages.ChangeInterfaceMessages.BecomeNinjaMsg
- Actorbase.Server.Core.Messages.ChangeInterfaceMessages.BecomeStoreKeeperMsg

4.16 Actorbase.Driver

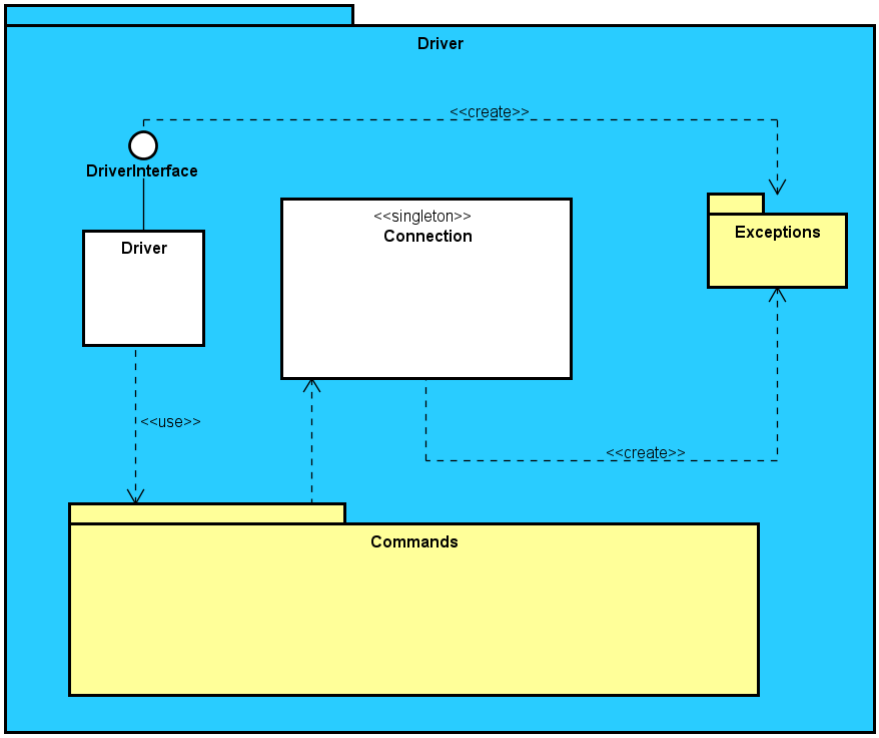


Figura 22: Componente Actorbase.Driver

4.16.1 Descrizione

Package per la componente Driver del sistema. Per la gestione dei comandi implementa il Design Pattern Command.

4.16.2 Package Figli

- Actorbase.Driver.Components
- Actorbase.Driver.Exceptions

4.16.3 Classi

- Actorbase.Driver.Driver
- Actorbase.Driver.Connection

4.16.4 Interfacce

- Actorbase.Driver.DriverInterface

4.17 Actorbase.Driver.Connection

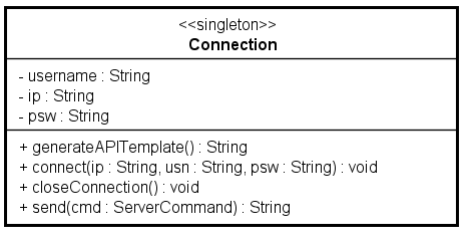


Figura 23: Classe Actorbase.Driver.Connection

4.17.1 Descrizione

Classe *singleton* che gestisce la comunicazione del Driver con il Server. Svolge il ruolo di *Receiver*.

4.17.2 Utilizzo

Viene usata per aprire e chiudere la connessione con il Server, inviare i messaggi e per generare la prima parte del comando per le API.

4.17.3 Relazione con altre classi

- **Actorbase.Driver.Commands.ServerCommand**: Relazione entrante, invio comando al Server.
- **Actorbase.Driver.Exception**: Relazione uscente, creazione di eccezioni.

4.18 Actorbase.Driver.Driver

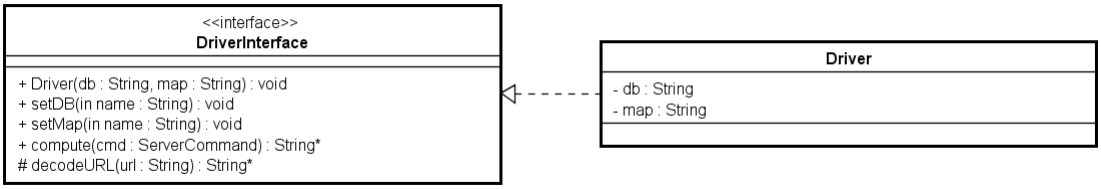


Figura 24: Classe Actorbase.Driver.Driver e interfaccia Actorbase.Driver.DriverInterface

4.18.1 Descrizione

Classe che gestisce la comunicazione mediante API. Svolge il ruolo di *Invoker*.

4.18.2 Utilizzo

Espone i metodi per codifica e decodifica dei comandi delle API. Inoltre mantiene le informazioni relative ad eventuali DataBase o Mappe selezionate.

4.18.3 Interfacce Estese

Actorbase.Driver.DriverInterface

4.18.4 Relazioni con altre classi

- **Actorbase.Driver.Commands**: Relazione uscente, utilizzo del metodo Execute.
- **Actorbase.Driver.Exceptions**: Relazione uscente, creazione di eccezioni.

4.19 Actorbase.Driver.Commands

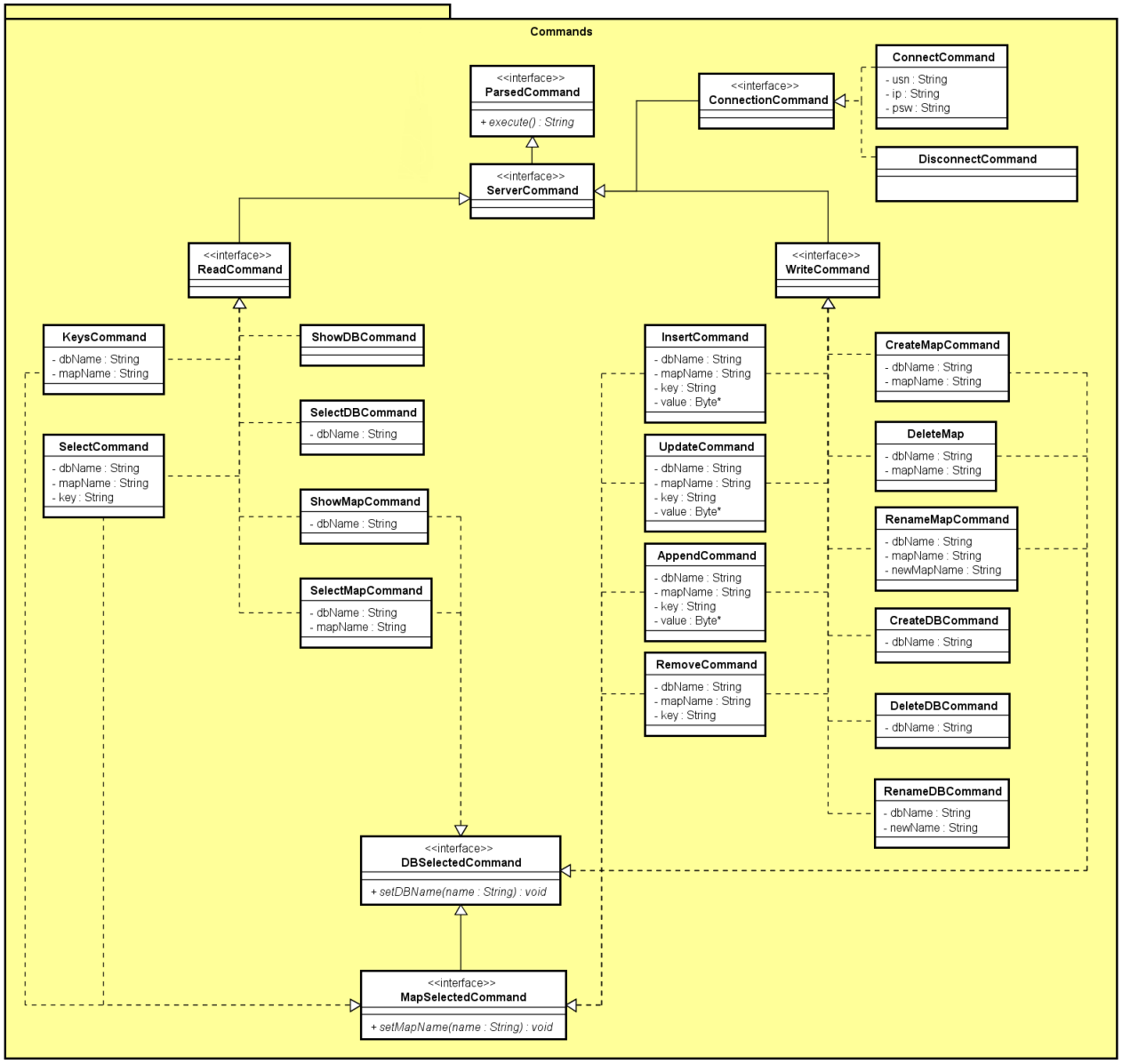


Figura 25: Package Actorbase.Driver.Commands

4.19.1 Descrizione

Package che contiene i la gerarchia di comandi dell'interfaccia *Command*. Ognuno di essi rappresenta astrattamente una operazione che deve essere svolta a livello del Server. Sono presenti due gerarchie di interfacce, una per la tipologia di comandi (lato client, lato server, lettura, scrittura) e l'altra per la tipologia di "preselezioni" necessarie per eseguire il comando (DataBase selezionato, Mappa selezionata).

4.19.2 Interfacce

- Actorbase.Driver.Commands.ServerCommand
- Actorbase.Driver.Commands.ReadCommand
- Actorbase.Driver.Commands.WriteCommand
- Actorbase.Driver.Commands.ConnectionCommand

- Actorbase.Driver.Commands.DBSelectedCommand
- Actorbase.Driver.Commands.MapSelectedCommand

4.19.3 Classi

- Actorbase.Driver.Commands.KeysCommand
- Actorbase.Driver.Commands.SelectCommand
- Actorbase.Driver.Commands.ShowBDCommand
- Actorbase.Driver.Commands.SelectDBCommand
- Actorbase.Driver.Commands.ShowMapCommand
- Actorbase.Driver.Commands.SelectMapCommand
- Actorbase.Driver.Commands.InsertCommand
- Actorbase.Driver.Commands.UpdateCommand
- Actorbase.Driver.Commands.AppendCommand
- Actorbase.Driver.Commands.RemoveCommand
- Actorbase.Driver.Commands.CreateMapCommand
- Actorbase.Driver.Commands.DeleteMapCommand
- Actorbase.Driver.Commands.RenameMapCommand
- Actorbase.Driver.Commands.CreateDBCommand
- Actorbase.Driver.Commands.DeleteDBCommand
- Actorbase.Driver.Commands.RenameDBCommand
- Actorbase.Driver.Commands.ConnectCommand
- Actorbase.Driver.Commands.DisconnectCommand

4.20 Actorbase.Driver.Commands.ParsedCommand

4.20.1 Descrizione

Interfaccia base che rappresenta un generico comando che il Driver sa trattare.

4.20.2 Interfacce Figle

Actorbase.Driver.Commands.ServerCommand

4.21 Actorbase.Driver.Commands.ServerCommand

4.21.1 Descrizione

Interfaccia base che rappresenta un comando lato Server.

4.21.2 Interfacce Estese

Actorbase.Driver.Commands.ParsedCommand

4.21.3 Interfacce Figlie

- Actorbase.Driver.Commands.ReadCommand
- Actorbase.Driver.Commands.WriteCommand
- Actorbase.Driver.Commands.ConnectionCommand

4.22 Actorbase.Driver.Commands.ReadCommand

4.22.1 Descrizione

Interfaccia base che rappresenta un comando per cui sono necessari i permessi di lettura sulla tabella selezionata

4.22.2 Interfacce Estese

Actorbase.Driver.Commands.ServerCommand

4.22.3 Classi Figlie

- **Actorbase.Driver.Commands.KeysCommand**: Classe concreta che rappresenta il comando per mostrare la lista delle chiavi nella mappa selezionata.
- **Actorbase.Driver.Commands.SelectCommand**: Classe concreta che rappresenta il comando per selezionare il valore di una chiave nella mappa selezionata.
- **Actorbase.Driver.Commands.ShowBDCommand**: Classe concreta che rappresenta il comando per mostrare l'elenco dei DataBase di cui si hanno permessi di lettura.
- **Actorbase.Driver.Commands.SelectDBCommand**: Classe concreta che rappresenta il comando per selezionare un determinato DataBase.
- **Actorbase.Driver.Commands.ShowMapCommand**: Classe concreta che rappresenta il comando per mostrare la lista delle mappe nel DataBase selezionato.
- **Actorbase.Driver.Commands.SelectMapCommand**: Classe concreta che rappresenta il comando per selezionatare una determinata mappa.

4.23 Actorbase.Driver.Commands.WriteCommand

4.23.1 Descrizione

Interfaccia base che rappresenta un comando per cui sono necessari i permessi di scrittura sulla tabella selezionata

4.23.2 Interfacce Estese

Actorbase.Driver.Commands.ServerCommand

4.23.3 Classi Figlie

- **Actorbase.Driver.Commands.InsertCommand**: Classe concreta che rappresenta il comando per inserire una coppia chiave-valore nella mappa selezionata.
- **Actorbase.Driver.Commands.UpdateCommand**: Classe concreta che rappresenta il comando per aggiornare il valore di una chiave nella mappa selezionata.
- **Actorbase.Driver.Commands.AppendCommand**: Classe concreta che rappresenta il comando per appendere in coda al valore di una chiave nella mappa selezionata.
- **Actorbase.Driver.Commands.RemoveCommand**: Classe concreta che rappresenta il comando per rimuovere una coppia chiave-valore dalla mappa selezionata.
- **Actorbase.Driver.Commands.CreateMapCommand**: Classe concreta che rappresenta il comando per creare una nuova mappa nel DataBase selezionato.
- **Actorbase.Driver.Commands.DeleteMapCommand**: Classe concreta che rappresenta il comando per eliminare una mappa nel DataBase selezionato.
- **Actorbase.Driver.Commands.RenameMapCommand**: Classe concreta che rappresenta il comando per rinominare una mappa nel DataBase selezionato.

- **Actorbase.Driver.Commands.CreateDBCommand**: Classe concreta che rappresenta il comando per creare un nuovo DataBase.
- **Actorbase.Driver.Commands.DeleteDBCommand**: Classe concreta che rappresenta il comando per rimuovere un DataBase.
- **Actorbase.Driver.Commands.RenameDBCommand**: Classe concreta che rappresenta il comando per rinominare un DataBase.

4.24 Actorbase.Driver.Commands.ConnectionCommand

4.24.1 Descrizione

Interfaccia base che rappresenta un comando di connessione o disconnessione dal Server.

4.24.2 Interfacce Estese

Actorbase.Driver.Commands.ServerCommand

4.24.3 Classi Figlie

- **Actorbase.Driver.Commands.ConnectCommand**: Classe concreta che rappresenta il comando per la connessione al server.
- **Actorbase.Driver.Commands.DisconnectCommand**: Classe concreta che rappresenta il comando per la disconnessione dal server.

4.25 Actorbase.Driver.Commands.DBSelectedCommand

4.25.1 Descrizione

Interfaccia base che rappresenta un generico comando che per cui è necessario avere un DataBase selezionato.

4.25.2 Classi Figlie

- Actorbase.Driver.Commands.ShowMapCommand
- Actorbase.Driver.Commands.SelectMapCommand
- Actorbase.Driver.Commands.CreateCommand
- Actorbase.Driver.Commands.DeleteCommand
- Actorbase.Driver.Commands.RenameCommand

4.26 Actorbase.Driver.Commands.MapSelectedCommand

4.26.1 Descrizione

Interfaccia base che rappresenta un generico comando che per cui è necessario avere una mappa selezionata.

4.26.2 Interfacce Estese

Actorbase.Driver.Commands.DBSelectedCommand

4.26.3 Classi Figlie

- Actorbase.Driver.Commands.KeysCommand
- Actorbase.Driver.Commands.SelectCommand
- Actorbase.Driver.Commands.InsertCommand
- Actorbase.Driver.Commands.UpdateCommand
- Actorbase.Driver.Commands.AppendCommand
- Actorbase.Driver.Commands.RemoveCommand

4.27 Actorbase.Client

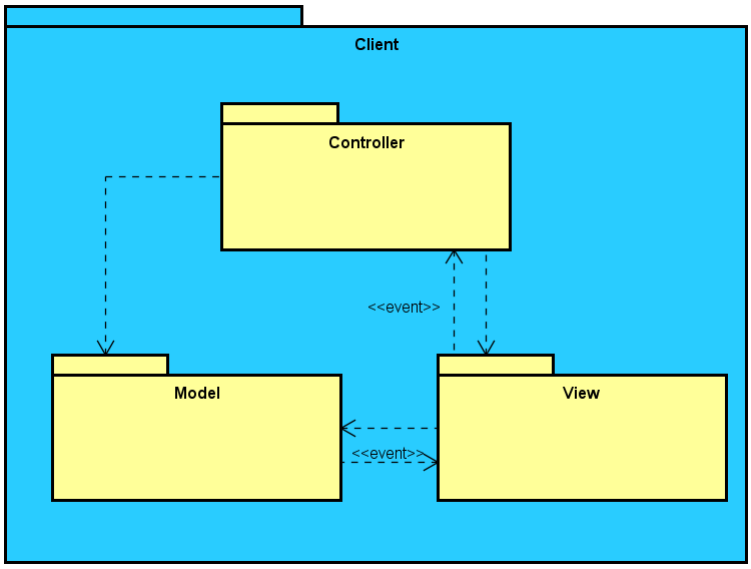


Figura 26: Package Actorbase.Client

4.27.1 Descrizione

Package per la componente Client del sistema. Implementa il Design Pattern Model View Controller.

4.27.2 Package Figli

- Actorbase.Client.Model
- Actorbase.Client.View
- Actorbase.Client.Controller

4.28 Actorbase.Client.Model

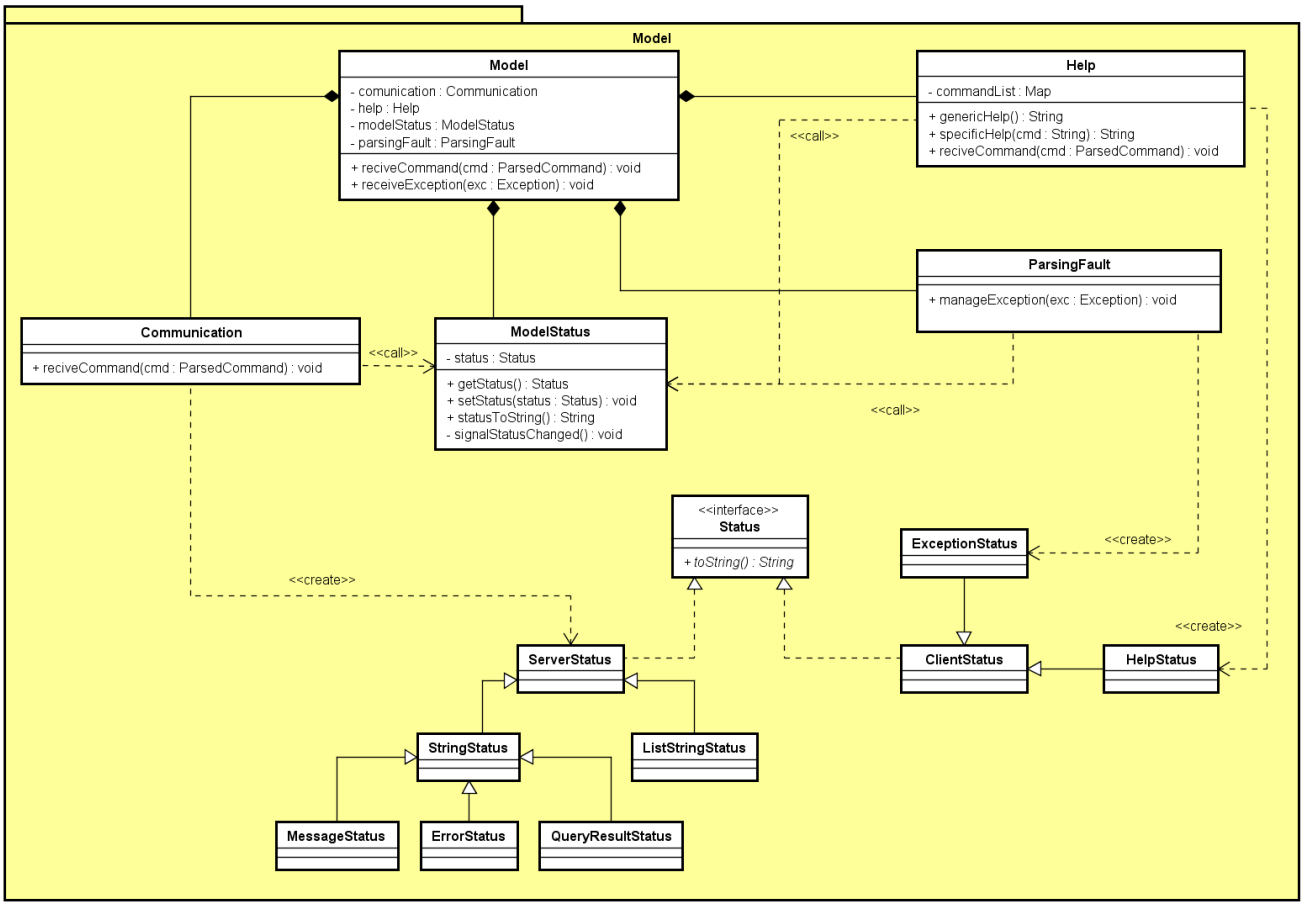


Figura 27: Package Actorbase.Client.Model

4.28.1 Descrizione

Package per la componente Model del pattern MVC della componente Client.

4.28.2 Interfacce

- Actorbase.Client.Model.Status

4.28.3 Classi

- Actorbase.Client.Model.Model
- Actorbase.Client.Model.Help
- Actorbase.Client.Model.ParsingFault
- Actorbase.Client.Model.Comunication
- Actorbase.Client.Model.ModelStatus
- Actorbase.Client.Model.ServerStatus
- Actorbase.Client.Model.StringStatus
- Actorbase.Client.Model.ListStringStatus
- Actorbase.Client.Model.MessageErrorStatus

- Actorbase.Client.Model.QueryResultStatus
- Actorbase.Client.Model.ExceptionStatus
- Actorbase.Client.Model.HelpStatus

4.29 Actorbase.Client.Model.Status

4.29.1 Descrizione

Interfaccia base della gerarchia degli stati del Model.

4.29.2 Utilizzo

Rappresenta un generico stato che il Model può assumere, è la rappresentazione astratta della frazione di stato dell'intero sistema Actorbase che è richiesta dall'utente. Le sue classi figlie espongono tutte un metodo per poter restituire in forma di stringa le informazioni alla View.

4.29.3 Classi Figlie

- Actorbase.Client.Model.ServerStatus
- Actorbase.Client.Model.ClientStatus

4.30 Actorbase.Client.Model.ServerStatus

4.30.1 Descrizione

Classe base per un generico stato restituito dal Server.

4.30.2 Utilizzo

Rappresenta un generico stato del Server che il Model può assumere.

4.30.3 Interfacce Estese

Actorbase.Client.Model.Status

4.30.4 Relazioni con altre classi

- **Actorbase.Client.Model.Communication**: Relazione entrante, creazione.

4.30.5 Classi Figlie

- **Actorbase.Client.Model.ListStringStatus**: Stato che rappresenta un generico stato di tipo lista di stringhe.
- **Actorbase.Client.Model.StringStatus**: Stato che rappresenta un generico stato di tipo stringa.
  - **Actorbase.Client.Model.MessageStatus**: Stato che rappresenta un messaggio del server.
  - **Actorbase.Client.Model.ErrorStatus**: Stato che rappresenta un errore a livello del server.
  - **Actorbase.Client.Model.QueryResultStatus**: Stato che rappresenta il risultato di una query ritornato in forma di stringa.

4.30.6 Actorbase.Client.Model.ClientStatus

4.30.7 Descrizione

Classe base per un generico stato restituito dal Client stesso.

4.30.8 Utilizzo

Rappresenta un generico stato del Client che il Model può assumere.

4.30.9 Interfacce Estese

Actorbase.Client.Model.Status

4.30.10 Relazioni con altre classi

- **Actorbase.Client.Model.Help**: Relazione entrante, creazione.
- **Actorbase.Client.Model.ParsingFault**: Relazione entrante, creazione.

4.30.11 Classi Figlie

- **Actorbase.Client.Model.ExceptionStatus**: Stato che rappresenta una eccezione.
- **Actorbase.Client.Model.HelpStatus**: Stato che rappresenta una richiesta di help da parte dell'utente.

4.31 Actorbase.Client.Model.Model

4.31.1 Descrizione

Classe base per la componente Model del Pattern MVC della componente Client.

4.31.2 Utilizzo

Fornisce un accesso unico al sottosistema delle classi ad esso composte.

4.31.3 Relazioni con altre classi

- **Actorbase.Client.Model.Communication**: Relazione uscente, composizione.
- **Actorbase.Client.Model.ModelStatus**: Relazione uscente, composizione.
- **Actorbase.Client.Model.Help**: Relazione uscente, composizione.
- **Actorbase.Client.Model.ParsingFault**: Relazione uscente, composizione.
- **Actorbase.Client.Controller**: Relazione entrante, chiamata metodo per la ricezione del comando.

4.32 Actorbase.Client.Model.Comunication

4.32.1 Descrizione

Classe che gestisce la comunicazione con il server tramite Driver.

4.32.2 Utilizzo

Gestisce la comunicazione con il server tramite Driver, crea gli ServerStatus e chiama il metodo per modificare l'attributo status del Model.

4.32.3 Relazioni con altre classi

- **Actorbase.Client.Model.Model**: Relazione entrante, composizione.
- **Actorbase.Client.Model.ServerStatus**: Relazione uscente, creazione.
- **Actorbase.Client.Model.ModelStatus**: Relazione uscente, impostazione dello status.
- **Actorbase.Driver.Driver**: Relazione uscente, composizione.

4.33 Actorbase.Client.Model.ModelStatus

4.33.1 Descrizione

Classe che gestisce l'attributo stato di Model.

4.33.2 Relazioni con altre classi

- **Actorbase.Client.Model.Model**: Relazione entrante, composizione.
- **Actorbase.Client.Model.Communication**: Relazione entrante, impostazione dello status.
- **Actorbase.Client.Model.Help**: Relazione entrante, impostazione dello status.
- **Actorbase.Client.Model.ParsingFault**: Relazione entrante, impostazione dello status.

4.34 Actorbase.Client.Model.Help

4.34.1 Descrizione

Classe che gestisce le richieste di aiuto.

4.34.2 Utilizzo

Gestisce le richieste di aiuto, crea gli HelpStatus e chiama il metodo per modificare l'attributo status del Model.

4.34.3 Relazioni con altre classi

- **Actorbase.Client.Model.Model**: Relazione entrante, composizione.
- **Actorbase.Client.Model.HelpStatus**: Relazione uscente, creazione.
- **Actorbase.Client.Model.ModelStatus**: Relazione uscente, impostazione dello status.

4.35 Actorbase.Client.Model.ParsingFault

4.35.1 Descrizione

Classe che gestisce le eccezioni.

4.35.2 Utilizzo

Gestisce le eccezioni, crea gli ExceptionStatus e chiama il metodo per modificare l'attributo status del Model.

4.35.3 Relazioni con altre classi

- **Actorbase.Client.Model.Model**: Relazione entrante, composizione.
- **Actorbase.Client.Model.ExceptionStatus**: Relazione uscente, creazione.
- **Actorbase.Client.Model.ModelStatus**: Relazione uscente, impostazione dello status.

4.36 Actorbase.Client.View

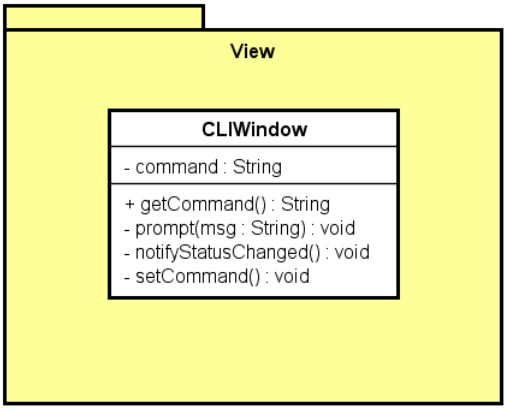


Figura 28: Package Actorbase.Client.View

4.36.1 Descrizione

Package per la componente View del pattern MVC della componente Client.

4.36.2 Classi

Actorbase.Client.View.View

4.37 Actorbase.Client.View.View

4.37.1 Descrizione

Classe che gestisce l'interfaccia utente.

4.37.2 Utilizzo

Gestisce l'interfaccia utente, permette di inserire comandi e stamparne l'output a video.

4.37.3 Relazioni con altre classi

- **Actorbase.Client.Controller.Controller**: Relazione entrante, richiesta comando inserito.
- **Actorbase.Client.Controller.Controller**: Relazione uscente, segnalazione di cambio di stato.
- **Actorbase.Client.Model.Model**: Relazione entrante, ricezione segnale di cambio di stato.
- **Actorbase.Client.Model.Model**: Relazione uscente, query sullo stato del Model.

5 Diagrammi delle attività

Segue la descrizione dei diagrammi delle attività che mostrano le possibili interazioni dell'utente con Actorbase. Il diagramma iniziale illustrerà le attività possibili che saranno successivamente mostrate in sotto diagrammi specifici, queste attività sono segnate nel diagramma principale con un fork.

5.0.1 Diagramma attività principale

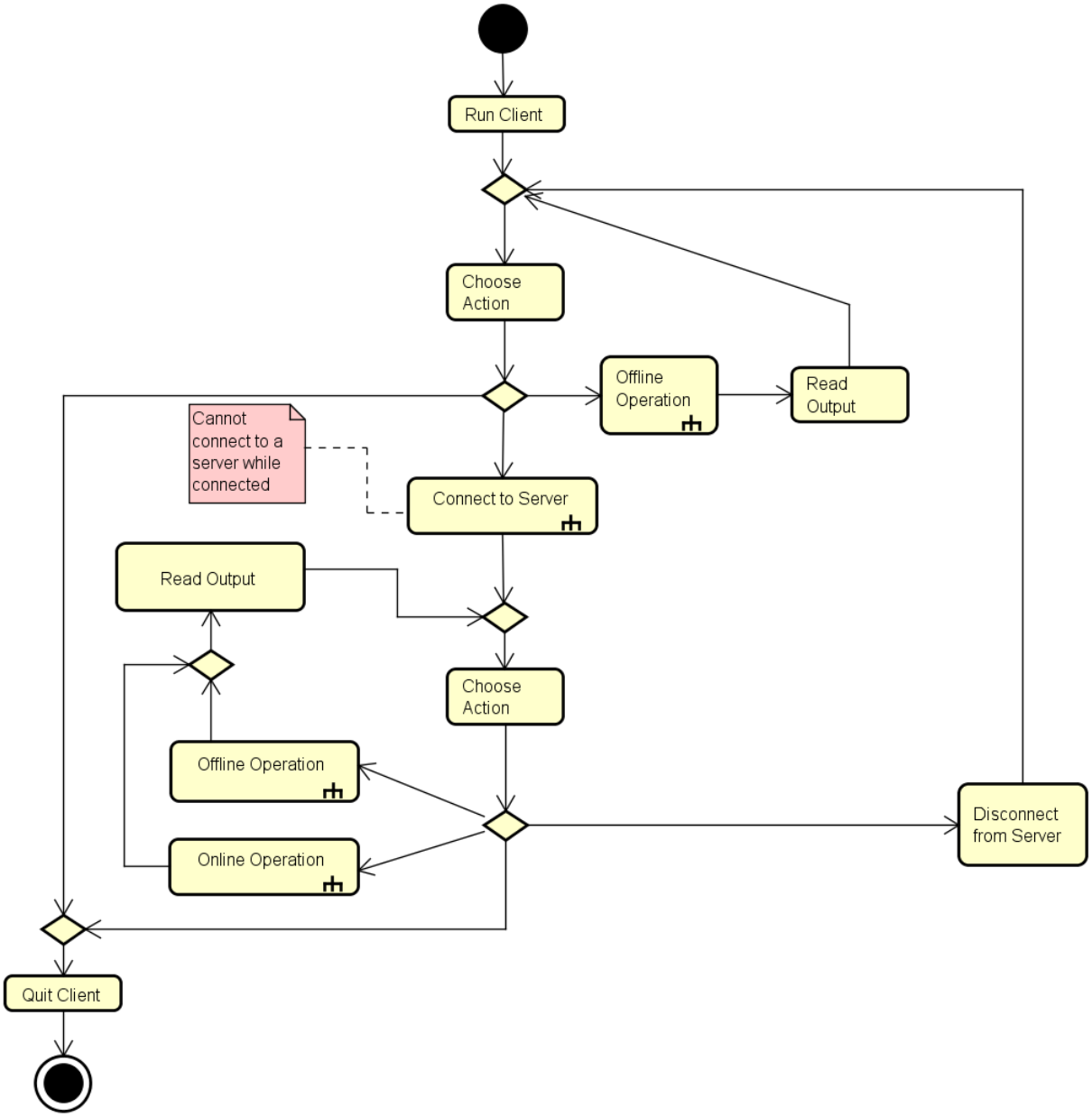


Figura 29: Diagramma attività principale

Dopo aver avviato il client l'utente può svolgere tre tipi di operazione: connettersi ad un server, chiudere l'applicativo o svolgere un operazione offline. Se sceglie di connettersi può sempre chiudere l'applicativo e svolgere operazioni che non necessitano di essere connessi al server, può in più disconnettersi o svolgere operazioni sul server, non potrà però più connettersi ad un server finché non effettua la disconnessione dal sever corrente.

5.0.2 Offline Operation

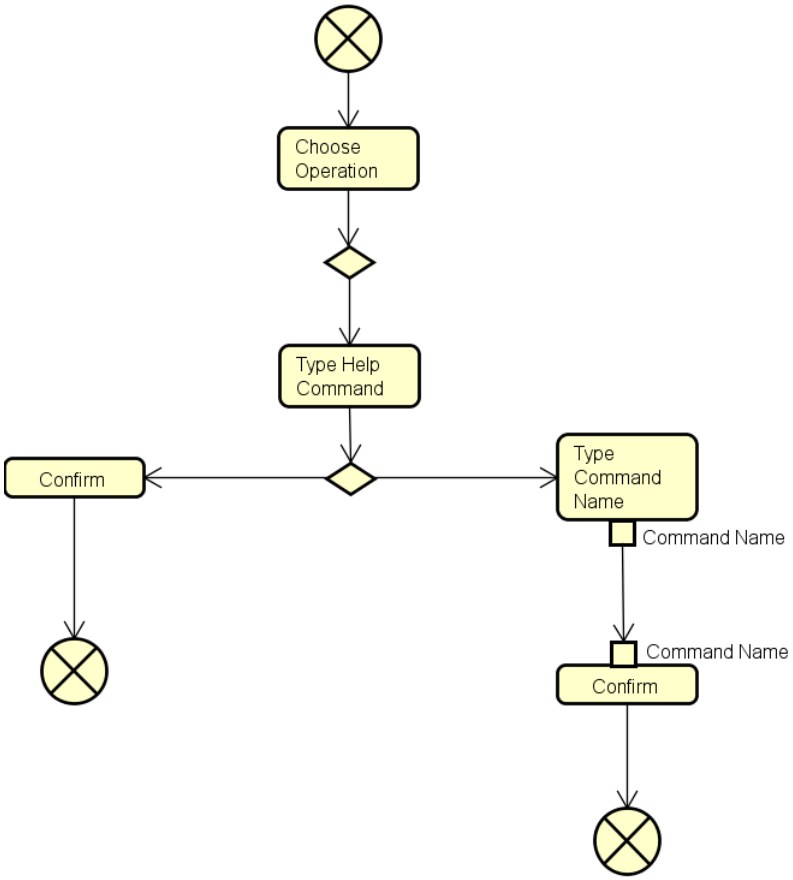


Figura 30: Diagramma attività operazioni offline

L'utente sceglie se chiedere un aiuto generale, o riguardante un comando specifico, nel primo caso è sufficiente che digiti il comando di aiuto e lo confermi, altrimenti dovrà successivamente scrivere il comando del quale vuole avere chiarimenti e poi confermare.

5.0.3 Connect to Server

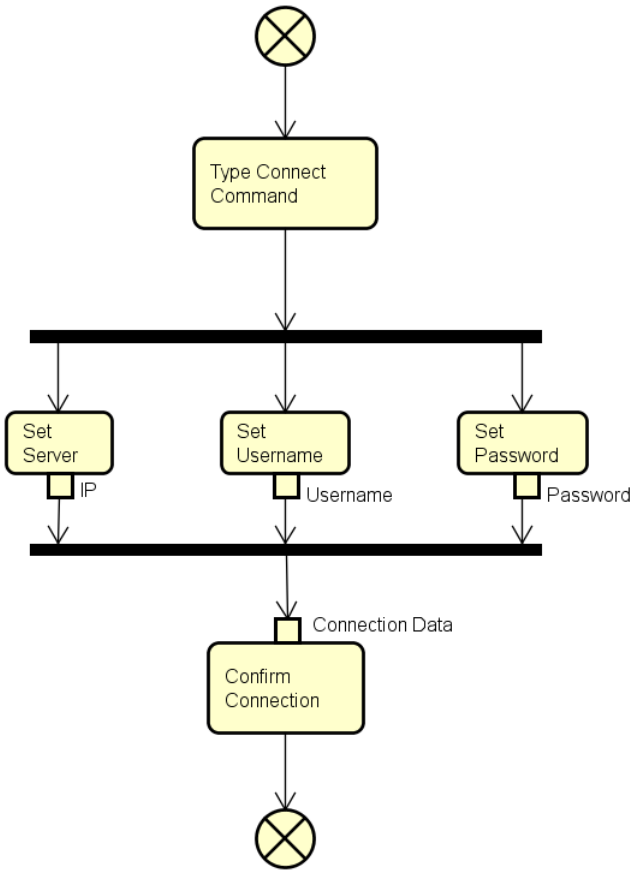


Figura 31: Diagramma attività connessione ad un server

Per effettuare una connessione l'utente dovrà digitare il commando di connessione e successivamente fornire i dati per la connessione, nello specifico: l'indirizzo IP al quale ci si vuole connettere, il nome utente e la password. Dopo di che è sufficiente che confermi il comando.

5.0.4 Online Operation

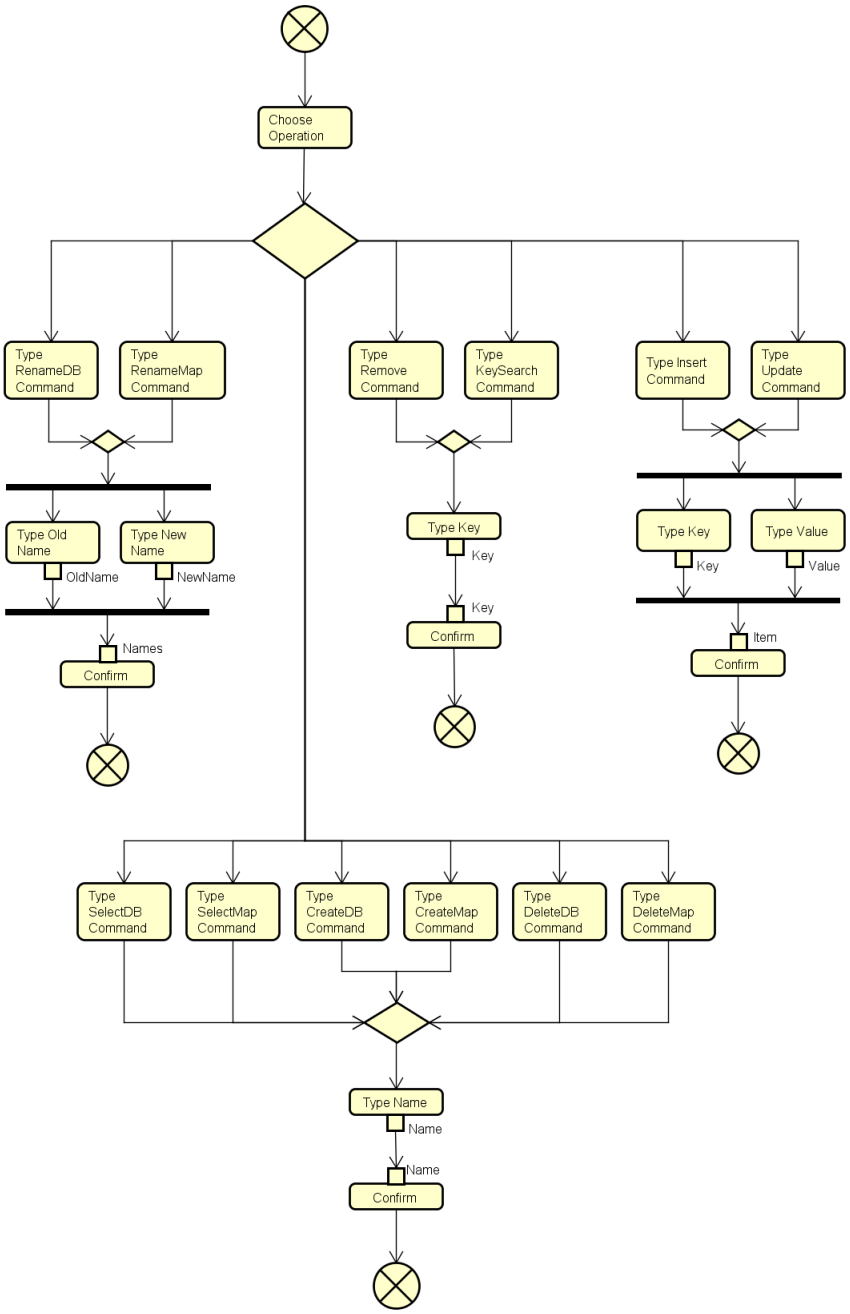


Figura 32: Diagramma attività operazioni online

L'utente sceglie che tipo di operazione vuole effettuare, se vuole effettuare un' operazione di rinomina su un database o una mappa è sufficiente che fornisca il comando di rinomina, il vecchio e il nuovo nome e che poi confermi l'operazione; se vuole effettuare un' operazione di selezione o di creazione o di cancellazione sia di un database che di una mappa, deve digitare il comando corretto e successivamente il nome del database o della mappa sulla quale vuole effettuare l'operazione dopo di che è sufficiente che confermi il comando; se vuole effettuare un' operazione di rimozione o selezione di un dato è necessario che digiti il comando desiderato, digiti la chiave completa e che confermi l'operazione; se vuole effettuare un inserimento di un dato o un aggiornamento di un dato deve inserire il comando corretto, digitare la chiave completa e confermare l'operazione.

6 Diagrammi di sequenza

In questa sezione verranno illustrati e descritti i principali diagrammi di sequenza realizzati. I diagrammi di sequenza realizzati illustrano come il lato server gestisce determinate richieste provenienti dall'esterno.

6.1 Avvio

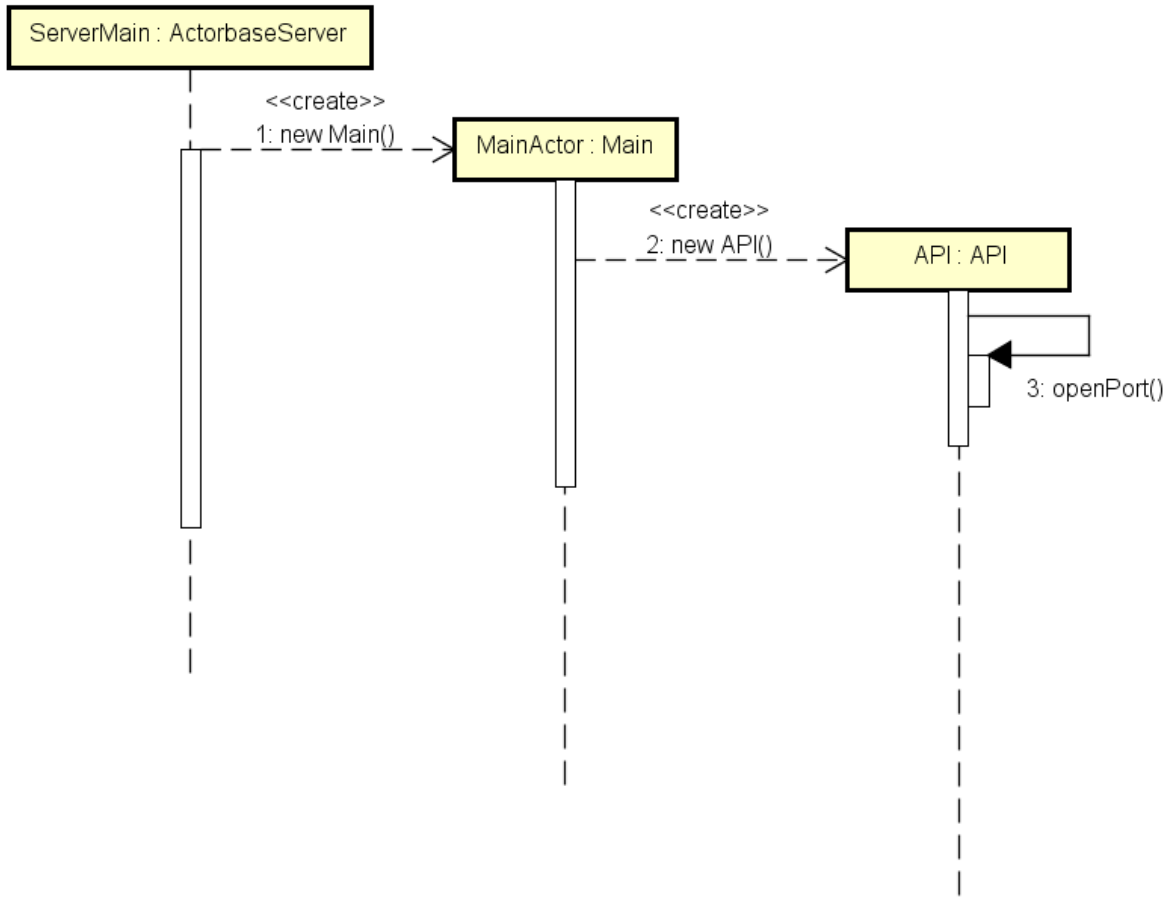


Figura 33: Diagramma di sequenza - avvio del server principale.

Quando viene avviata la parte server, viene creato un oggetto di classe **Main**. La classe in questione crea un oggetto a sua volta, di classe **API** che apre una porta per restare in ascolto di richieste di connessioni proveniente da un client.

6.2 Chiusura

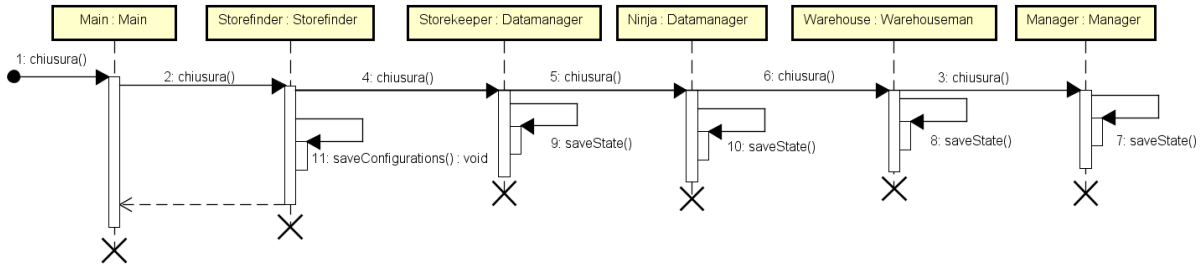


Figura 34: Diagramma di sequenza - chiusura del server.

Una volta inserito il comando di chiusura sulla shell del server, l'attore principale invia un messaggio a tutti i suoi figli, richiedendo la loro chiusura. Gli **Storefinder** a loro volta, prima di effettuare il salvataggio delle configurazioni inviano un messaggio a tutti i loro figli: **Manager**, **Storekeeper**, **Ninja**, **Warehouseman**. Questi ultimi effettuano tutti le loro operazioni di chiusura e salvataggio, e poi si interrompono. Una volta chiusi tutti i figli, l'attore principale può finalmente chiudersi a sua volta.

6.3 Richiesta esterna

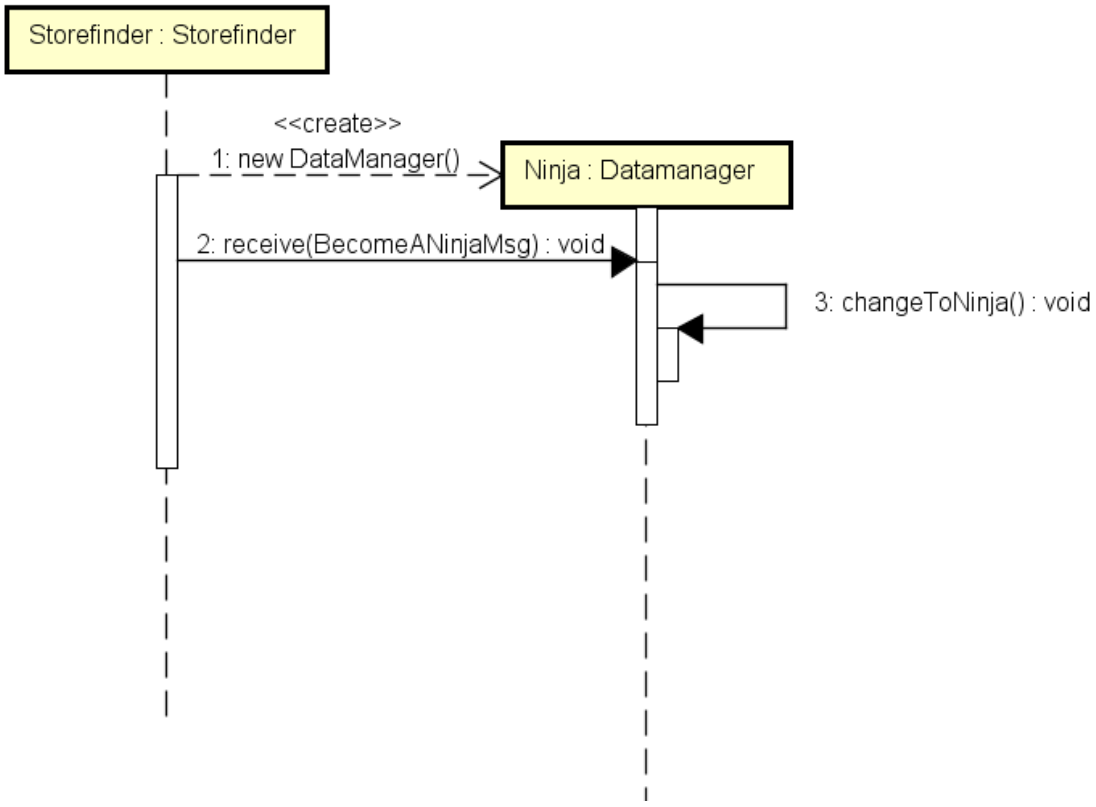


Figura 35: Diagramma di sequenza - gestione di richiesta esterna ad alto livello.

In questo diagramma è possibile visuallizzare le principali funzioni che vengono chiamate per la gestione di una richiesta esterna. L'attore principale, una volta ricevuto un messaggio HTTP dalle API, lo inoltra ad un **MessagesBuilder**, lo gestisce (nei diagrammi seguenti verrà spiegato meglio nel dettaglio), e

quando riceve una risposta, la inoltra ad un attore di tipo **HttpBuilder** che ricostruisce un messaggio HTTP da inoltrare alle API che a loro volta inoltreranno all'esterno.

6.4 Richiesta di creazione DB

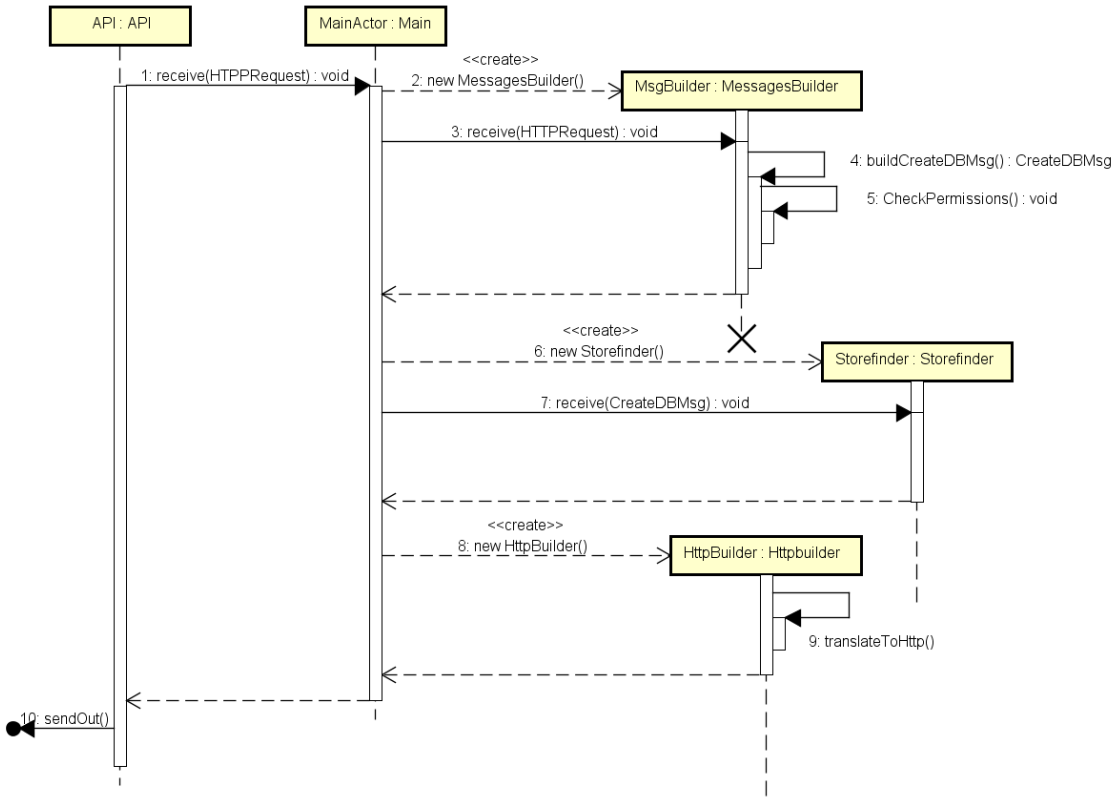


Figura 36: Diagramma di sequenza - gestione di richiesta di creazione DB.

In questo diagramma è possibile visualizzare quali attori vengono coinvolti nella creazione di un nuobo DB. Come in ogni richiesta, l'attore principale interpreta un **MessagesBuilder** prima di inoltrare il messaggio ai suoi attori figli, e un **HttpBuilder** prima di inviare la risposta alle **API**. Per la creazione di un DB, il **Main** non fa altro che creare un attore di tipo **Storefinder** per il DB desiderato.

6.5 Richiesta di find

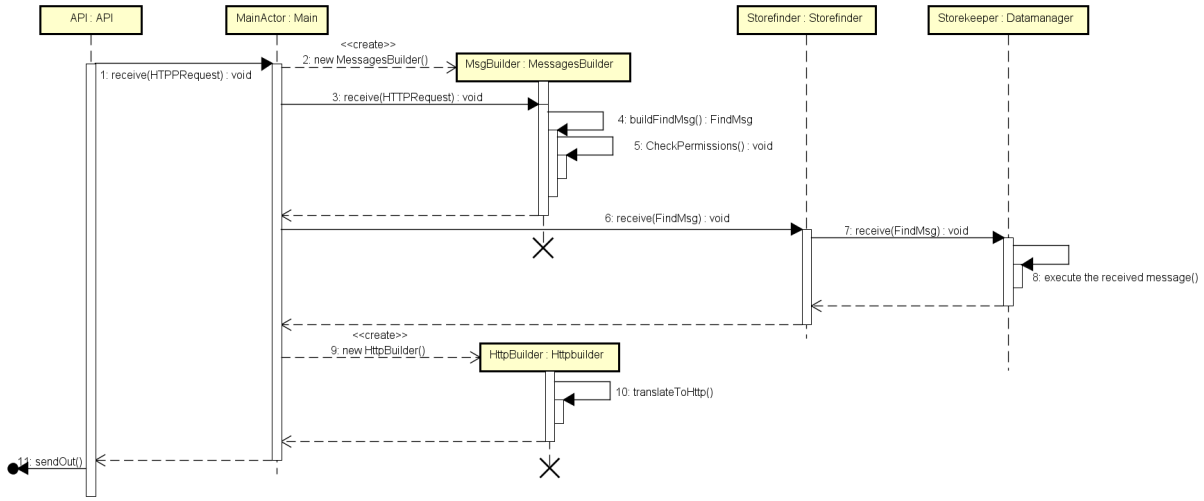


Figura 37: Diagramma di sequenza - gestione di richiesta di Find.

In questo diagramma, vengono mostrate le chiamate necessarie per gestire una richiesta Find proveniente da un client. Una volta arrivata allo **Storefinder**, esso inoltra la richiesta allo **Storekeeper** corretto, che interpreta la sua mappa interna e restituisce la risposta. Come nel diagramma generale, l'attore principale, prima di inviare la rispota alle **API** crea un **HttpBuilder** che trasforma la risposta in un messaggio HTTP.

6.6 Richiesta di aggiornamento item

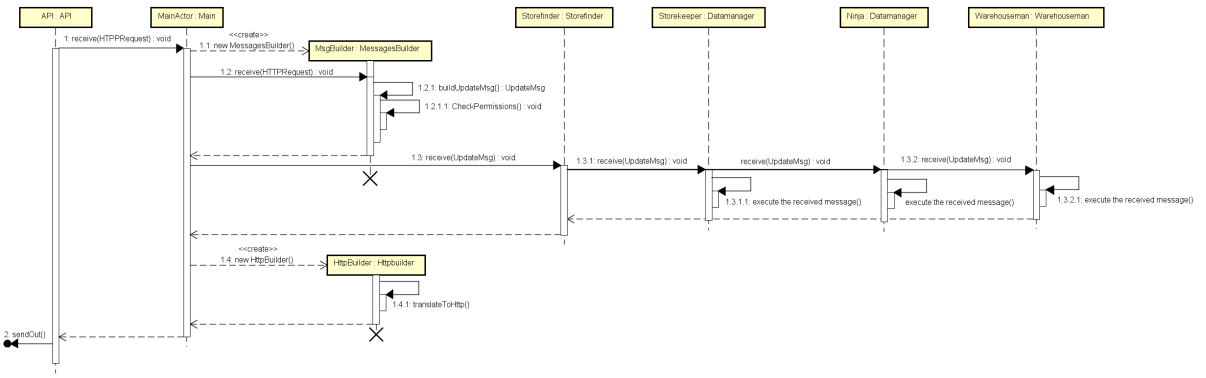


Figura 38: Diagramma di sequenza - gestione di richiesta di aggiornamento item.

Allo stesso modo della gestione di una richiesta di Find, il messaggio parte dall'attore principale ed arriva allo **Storekeeper** corretto. Quando riceve il messaggio di aggiornamento esso inoltra il messaggio di modifica ai suoi attori **Ninja** e **Warehouseman** e successivamente aggiorna la sua mappa interna. Il **Ninja** aggiorna a sua volta la sua mappa interna, mentre il **Warehouseman** effettua i cambiamenti su disco. Entrambi mandano una conferma allo **Storekeeper**, che a sua volta risponde allo **Storefinder**.

Allo stesso modo delle richieste generali, l'attore principale riceve la risposta dallo **Storefinder**, crea un **HttpBuilder** e poi manda la risposta alle **API**.

6.7 Creazione Ninja

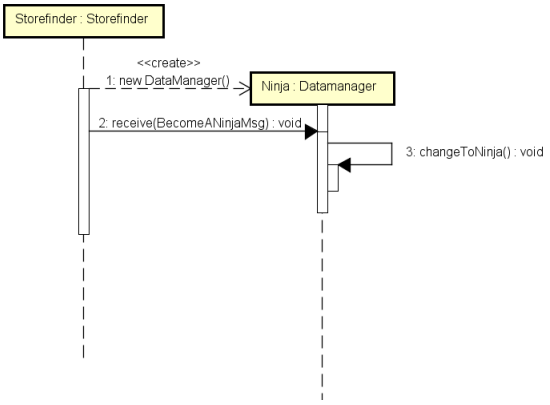


Figura 39: Diagramma di sequenza - creazione di un attore Ninja.

Per creare un attore di tipo Ninja, uno **Storefinder** crea un attore di classe **Datamanager** e lo converte immediatamente in un Ninja, inviandogli il messaggio apposito per la sua trasformazione in Ninja.

6.8 Creazione Ninja

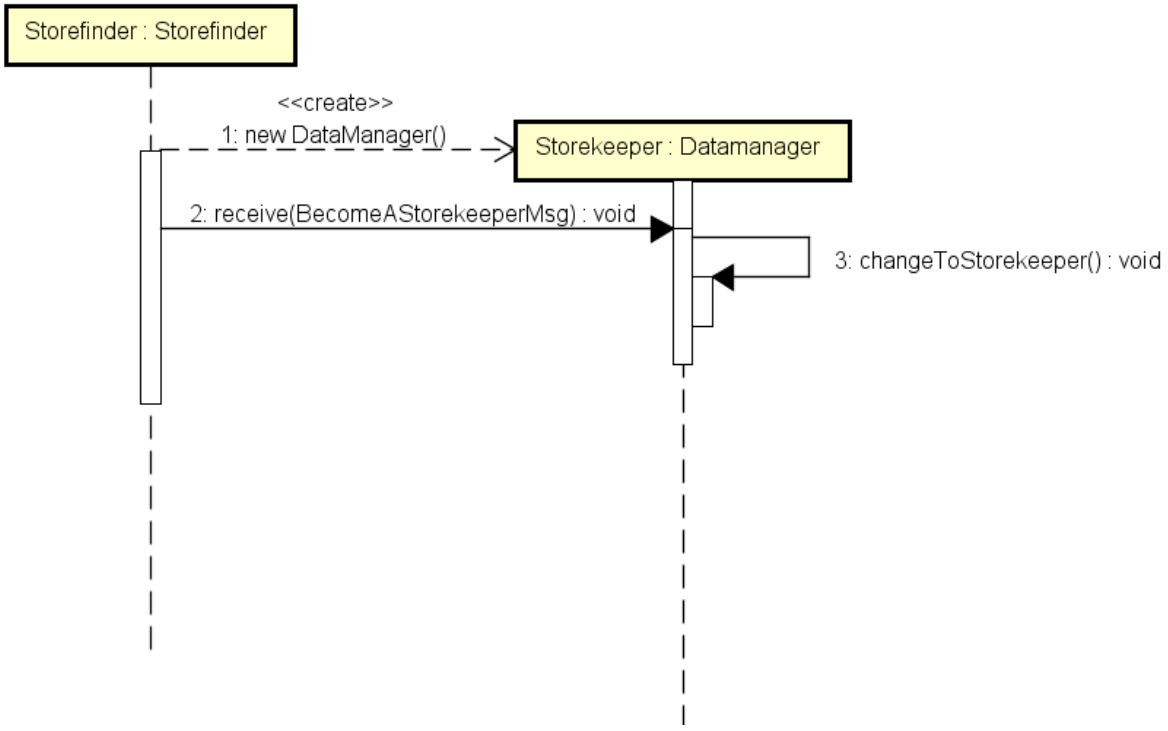


Figura 40: Diagramma di sequenza - creazione di un attore Storekeeper.

Allo stesso modo, per la creazione di uno **Storekeeper**, uno **Storefinder** crea un attore di classe **Datamanger** e ne cambia l'interfaccia in quella di uno **Storekeeper** inviandogli il messaggio apposito.

6.9 Sostituzione di uno Storekeeper

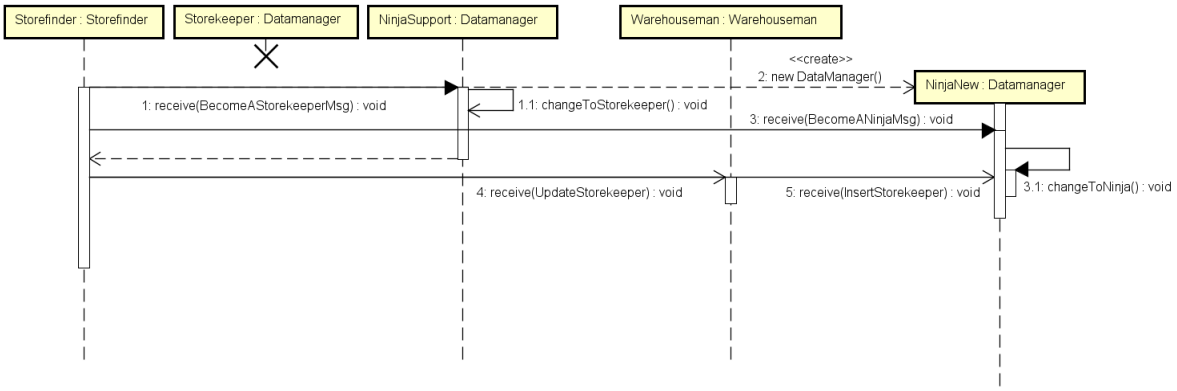


Figura 41: Diagramma di sequenza - sostituzione di un attore Storekeeper.

In questo diagramma è possibile visualizzare in che modo un attore **Storekeeper** che si arresta inaspettatamente viene sostituito dal suo **Ninja** di supporto. Come è possibile vedere nell'immagine, lo **Storefinder** manda un messaggio al **Ninja** per promuoverlo a **Storekeeper**. Successivamente crea un nuovo attore di tipo **Datamanager** e cambia la sua interfaccia in quella di un **Ninja**. Subito dopo notifica il nuovo **Ninja** e il **Warehouseman** del cambiamento.

7 Stime di fattibilità e di bisogno di risorse

L'architettura definita fino a questo punto è sufficiente per fornire una stima della fattibilità del prodotto e delle risorse richieste per la realizzazione.
Il gruppo inizialmente non aveva conoscenze sufficienti per stimare in modo appropriato la complessità dell'implementazione di un database basato sulla logica ad attori. Grazie al livello di dettaglio raggiunto sono stati fugati molti dei dubbi e delle incertezze a riguardo, confermando le previsioni sull'esito positivo del progetto.
Sono state inoltre individuate con chiarezza le risorse tecnologiche che verranno utilizzate:

- Akka: libreria per modello ad attori.
- IntelliJ: framework per la stesura del codice.
- JVM: piattaforma per il funzionamento di Scala.

Il gruppo in contemporanea si è dedicato allo studio delle nuove tecnologie raggiungendo un buon livello di conoscenza. L'insieme di queste risorse potrà garantire la realizzazione di tutte le componenti dell'architettura.

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

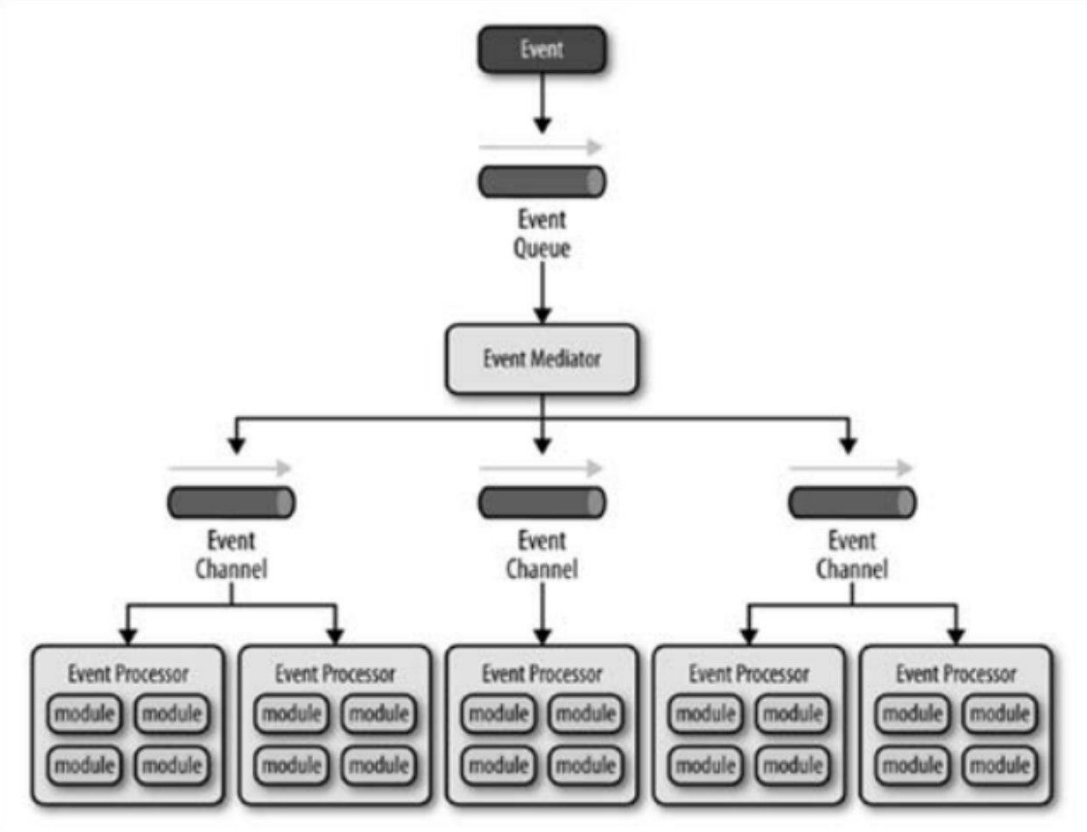


Figura 42: Diagramma del Desing Pattern Event-driven

- **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

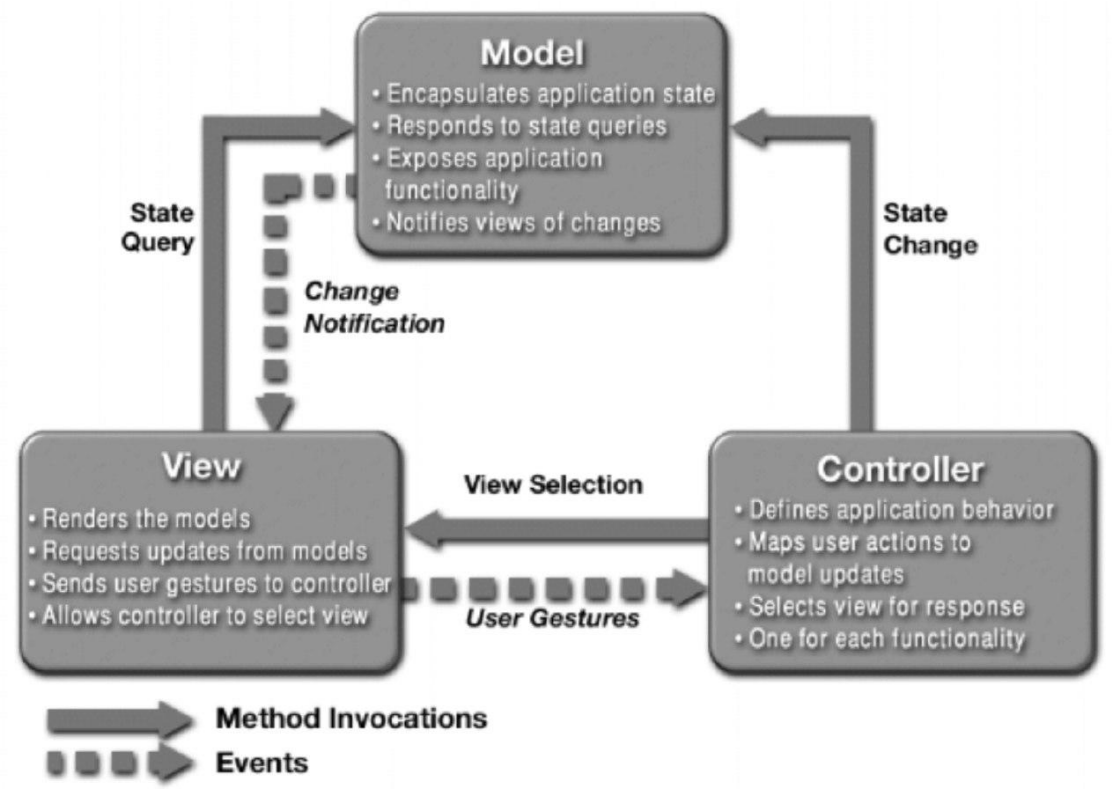


Figura 43: Diagramma del Desing Pattern MVC

- **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

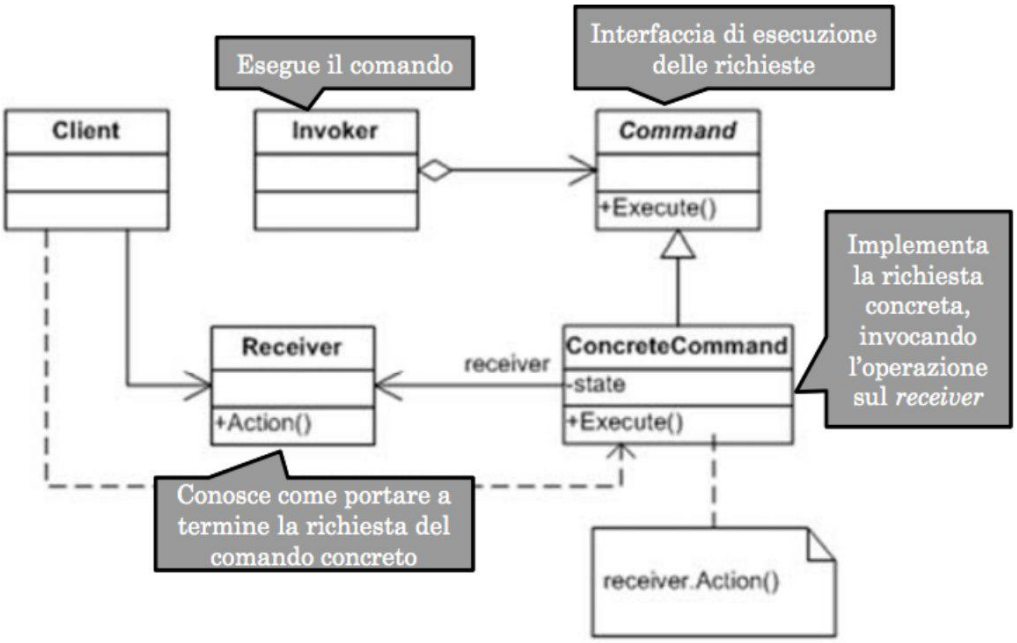


Figura 44: Diagramma del Desing Pattern Command

- **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

9.1.4 Singleton

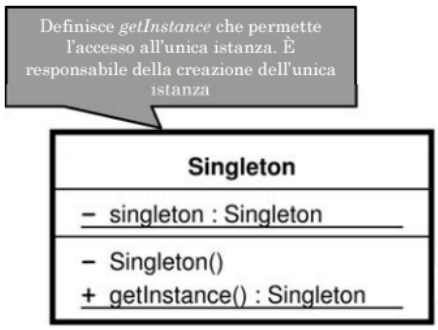


Figura 45: Diagramma del Desing Pattern Command

- **Scopo:** Assicurare che una classe abbia una sola istanza con un unico punto di accesso globale.

- **Motivazione:**È necessario assicurare che esista una sola istanza di alcune classi. Una classe Singleton ha la responsabilità sulle proprie istanze, in modo che nessuna altra istanza possa essere creata, e fornisce un punto di accesso unico.
- **Applicabilità:**
  - Deve esistere una ed una sola istanza di una classe in tutta l'applicazione, accessibile dai client in modo noto.
  - L'istanza deve essere estendibile con ereditarietà, consentendo ai client di non modificare il proprio codice.

9.1.5 Singleton

- **Scopo:** Assicurare che una classe abbia una sola istanza con un unico punto di accesso globale.
- **Motivazione:**È necessario assicurare che esista una sola istanza di alcune classi. Una classe Singleton ha la responsabilità sulle proprie istanze, in modo che nessuna altra istanza possa essere creata, e fornisce un punto di accesso unico.
- **Applicabilità:**
  - Deve esistere una ed una sola istanza di una classe in tutta l'applicazione, accessibile dai client in modo noto.
  - L'istanza deve essere estendibile con ereditarietà, consentendo ai client di non modificare il proprio codice.

Elenco delle figure

1 Scala - logo . . . 7
2 Akka - logo . . . 7
3 Architettura generale, vista Package . . . 8
4 Legenda . . . 8
5 Server, vista Package . . . 9
6 Architettura generale Client e Driver . . . 10
7 Componente Actorbase . . . 11
8 Componente Actorbase.Server . . . 12
9 Componente Actorbase.Server.API . . . 12
10 Componente Actorbase.Server.Core . . . 13
11 Componente Actorbase.Server.Core.Actors . . . 13
12 Componente Actorbase.Server.Core.Actors.DataManagement . . . 14
13 Componente Actorbase.Server.Core.Actors.Manager . . . 15
14 Componente Actorbase.Server.Core.Actors.StoreFinder . . . 15
15 Componente Actorbase.Server.Core.Messages . . . 16
16 Componente Actorbase.Server.Core.Messages.ConfigurationMessages . . . 17
17 Componente Actorbase.Server.Core.Messages.PermissionMessages . . . 18
18 Componente Actorbase.Server.Core.Messages.LinkActorsMessages . . . 18
19 Componente Actorbase.Server.Core.Messages.MainOperationMessages . . . 19
20 Componente Actorbase.Server.Core.Messages.DataManagerOperationMessages . . . 20
21 Componente Actorbase.Server.Core.Messages.ChangeInterfaceMessages . . . 20
22 Componente Actorbase.Driver . . . 21
23 Classe Actorbase.Driver.Connection . . . 22
24 Classe Actorbase.Driver.Driver e interfaccia Actorbase.Driver.DriverInterface . . . 22
25 Package Actorbase.Driver.Commands . . . 23
26 Package Actorbase.Client . . . 27
27 Package Actorbase.Client.Model . . . 28
28 Package Actorbase.Client.View . . . 32
29 Diagramma attività principale . . . 33
30 Diagramma attività operazioni offline . . . 34
31 Diagramma attività connessione ad un server . . . 35
32 Diagramma attività operazioni online . . . 36
33 Diagramma di sequenza - avvio del server principale. . . . 37
34 Diagramma di sequenza - chiusura del server. . . . 38
35 Diagramma di sequenza - gestione di richiesta esterna ad alto livello. . . . 38
36 Diagramma di sequenza - gestione di richiesta di creazione DB. . . . 39
37 Diagramma di sequenza - gestione di richiesta di Find. . . . 40
38 Diagramma di sequenza - gestione di richiesta di aggiornamento item. . . . 40
39 Diagramma di sequenza - creazione di un attore Ninja. . . . 41
40 Diagramma di sequenza - creazione di un attore Storekeeper. . . . 41
41 Diagramma di sequenza - sostituzione di un attore Storekeeper. . . . 42
42 Diagramma del Desing Pattern Event-driven . . . 45
43 Diagramma del Desing Pattern MVC . . . 46
44 Diagramma del Desing Pattern Command . . . 47
45 Diagramma del Desing Pattern Command . . . 47

Elenco delle tabelle

1 Diario delle modifiche 5