SWEeneyThreads

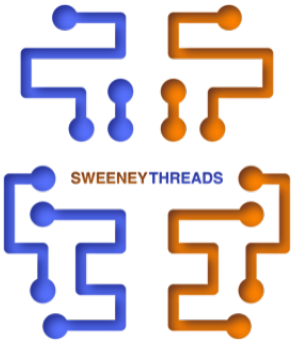
Actorbase

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 0.0.5

8 aprile 2016

Indice

Diario delle modifiche

Versione	Data	Autore	Descrizione
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/`

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.

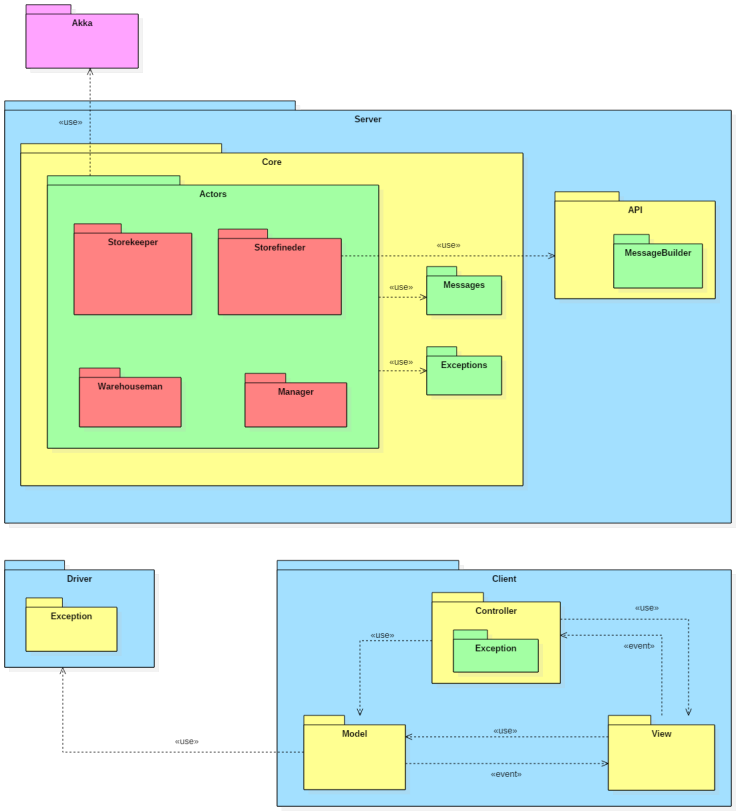


Figura 3: Architettura generale, vista Package

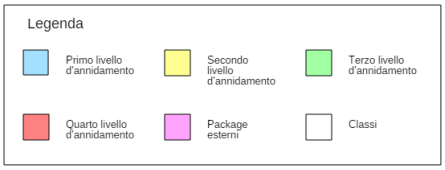


Figura 4: Legenda

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.

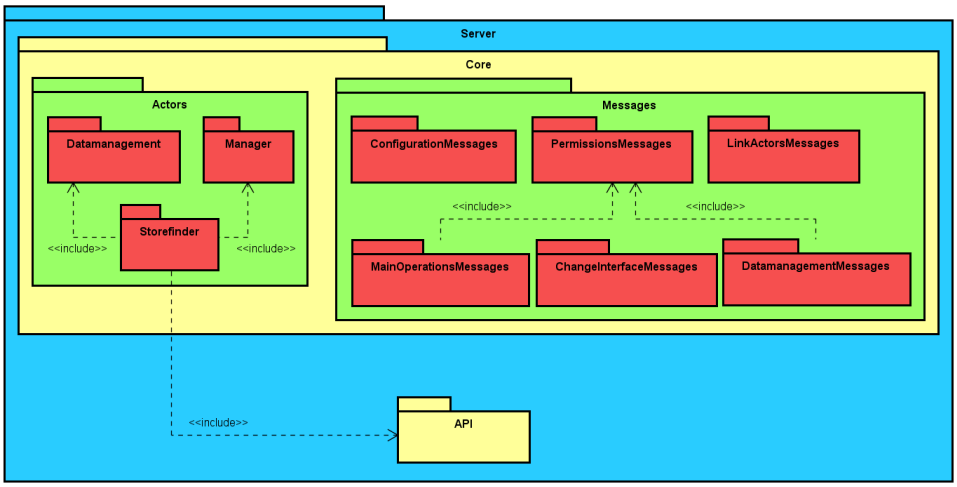


Figura 5: Server, vista Package

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.

4 Componenti

4.1 Actorbase

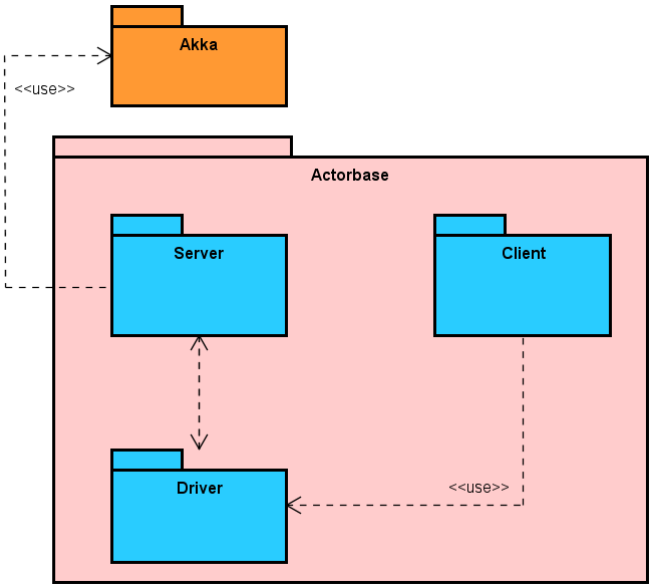


Figura 6: Componente Actorbase

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

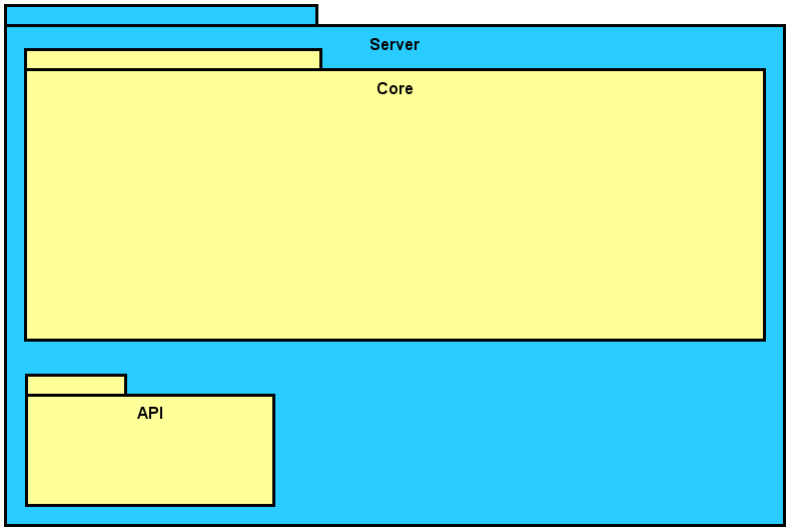


Figura 7: Componente Actorbase.Server

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

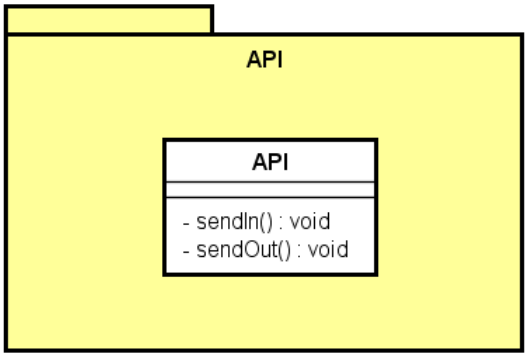


Figura 8: Componente Actorbase.Server.API

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

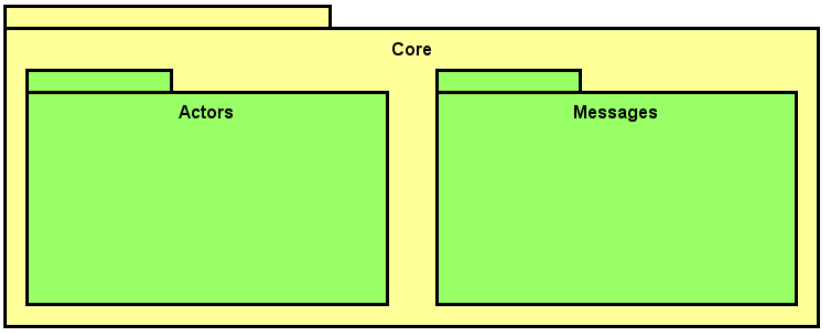


Figura 9: Componente Actorbase.Server.Core

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

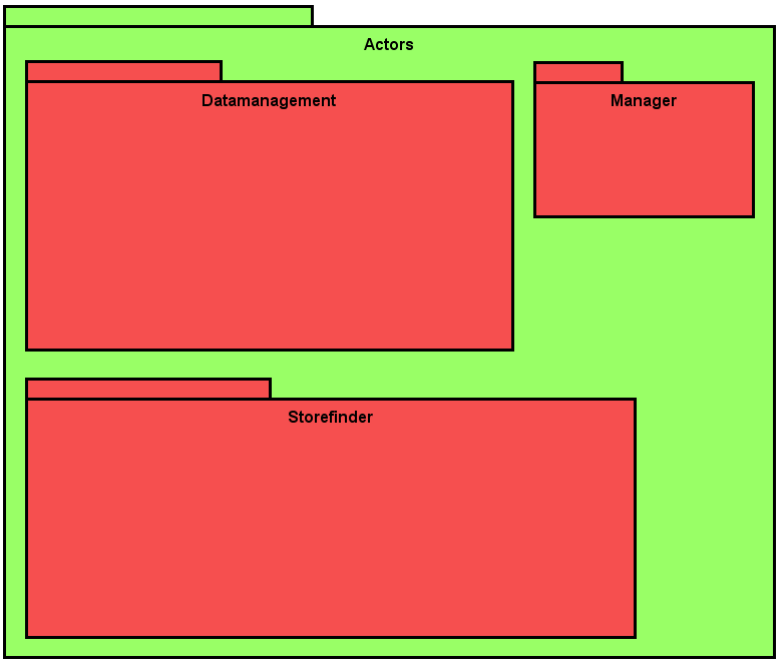


Figura 10: Componente Actorbase.Server.Core.Actors

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

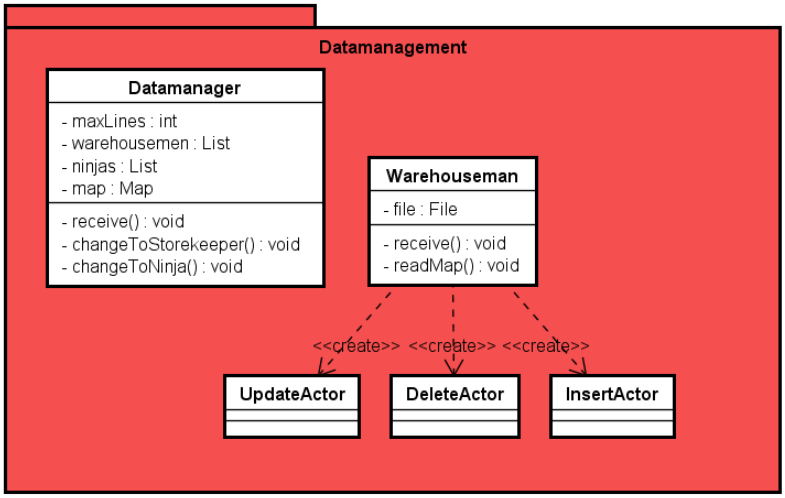


Figura 11: Componente Actorbase.Server.Core.Actors.DataManagement

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

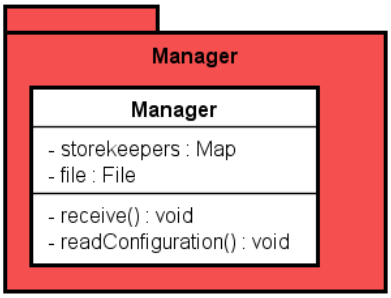


Figura 12: Componente Actorbase.Server.Core.Actors.Manager

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

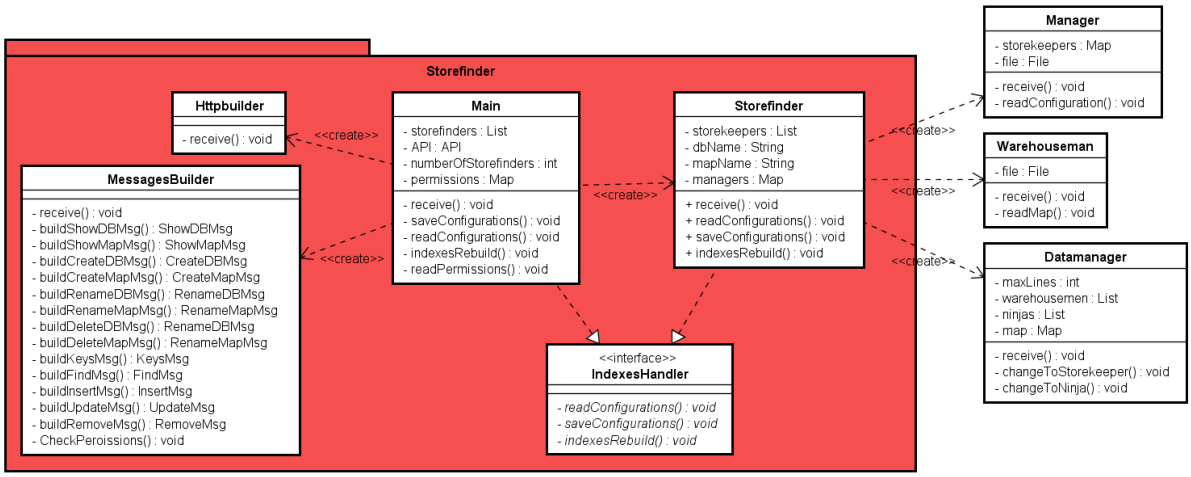


Figura 13: Componente Actorbase.Server.Core.Actors.StoreFinder

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

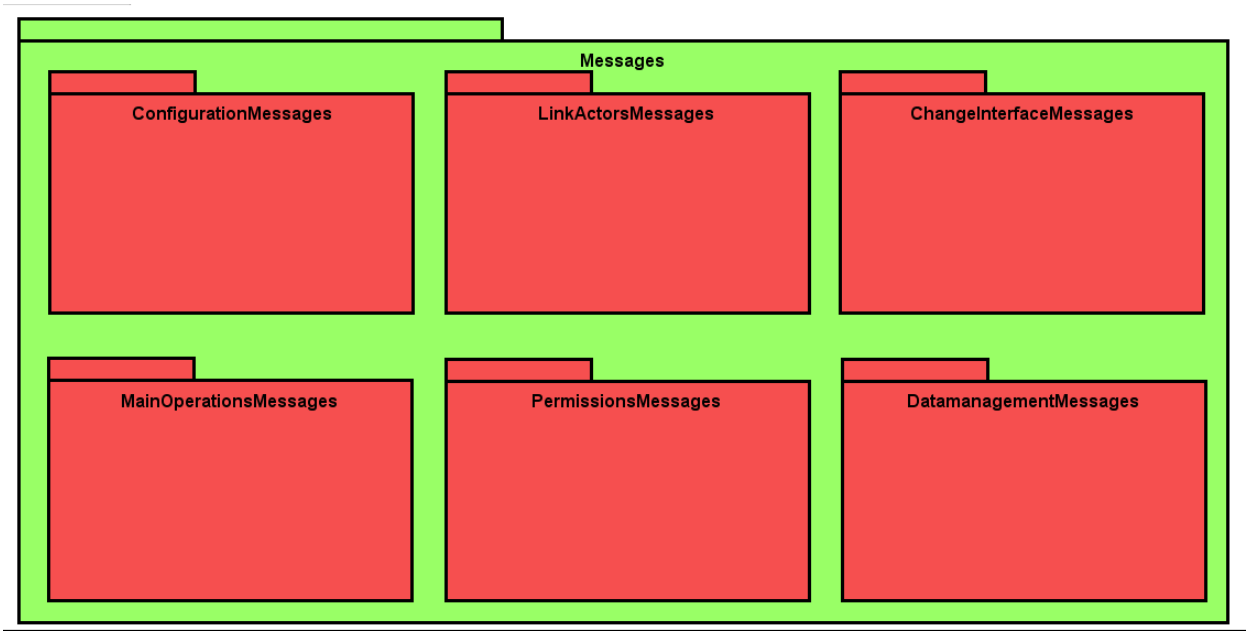


Figura 14: Componente Actorbase.Server.Core.Messages

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

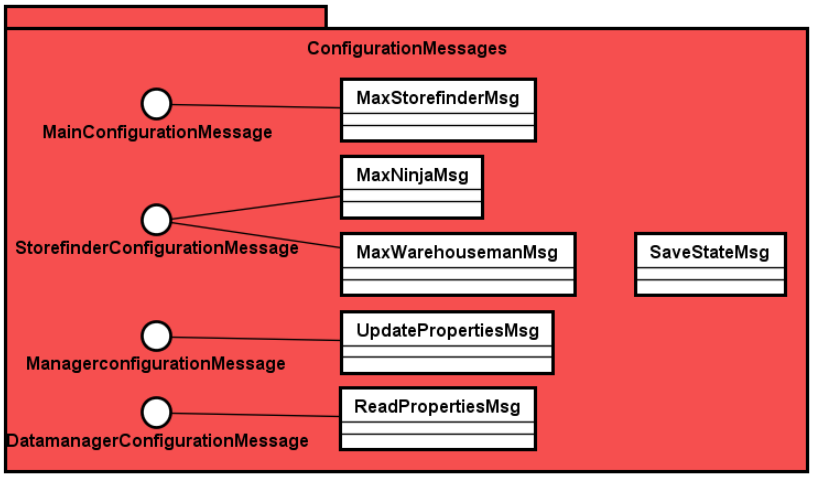


Figura 15: Componente Actorbase.Server.Core.Messages.ConfigurationMessages

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

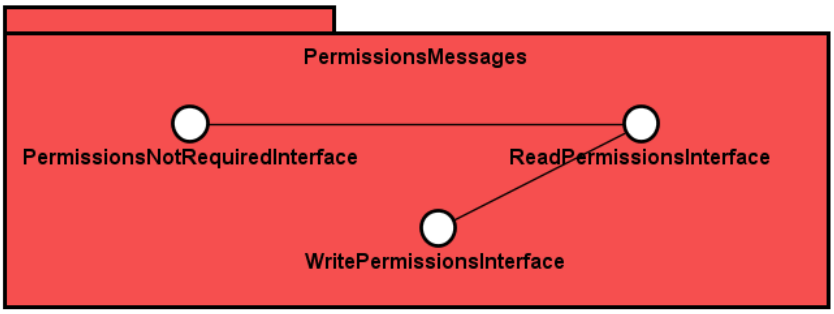


Figura 16: Componente Actorbase.Server.Core.Messages.PermissionMessages

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

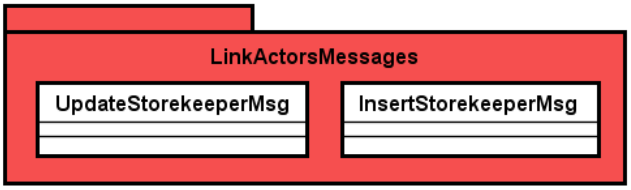


Figura 17: Componente Actorbase.Server.Core.Messages.LinkActorsMessages

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

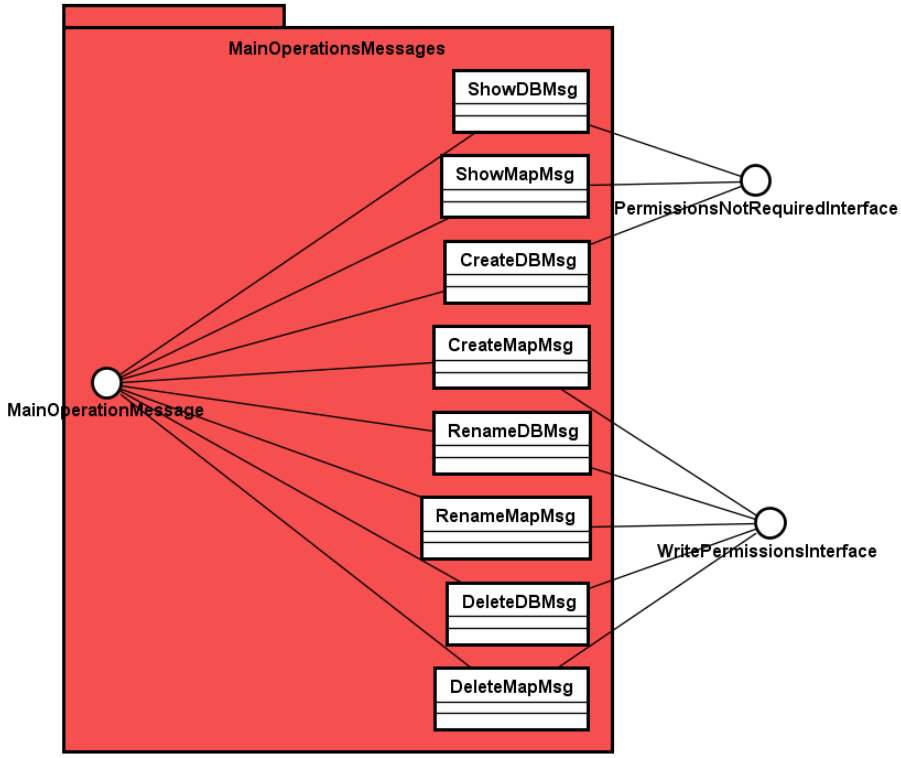


Figura 18: Componente Actorbase.Server.Core.Messages.MainOperationMessages

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

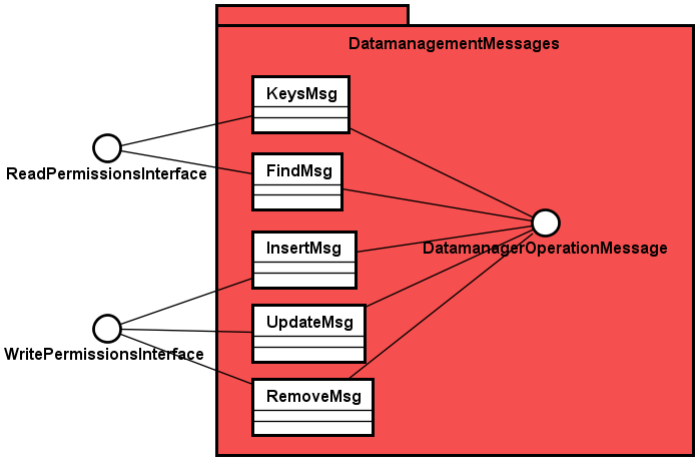


Figura 19: Componente Actorbase.Server.Core.Messages.DataManagerOperationMessages

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

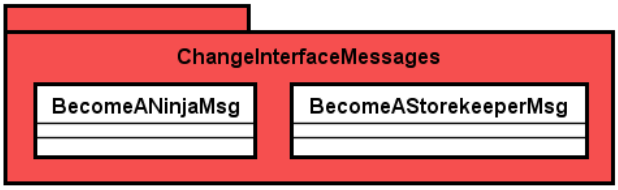


Figura 20: Componente Actorbase.Server.Core.Messages.ChangeInterfaceMessages

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

5 Classi

6 Diagrammi delle attività

7 Design pattern

8 Stime di fattibilità e di bisogno di risorse

9 Tracciamento

9.1 Tracciamento componenti-requisiti

9.2 Tracciamento requisiti-componenti

10 Appendice

10.1 Descrizione Desing Pattern

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

10.1.1 Event-driven

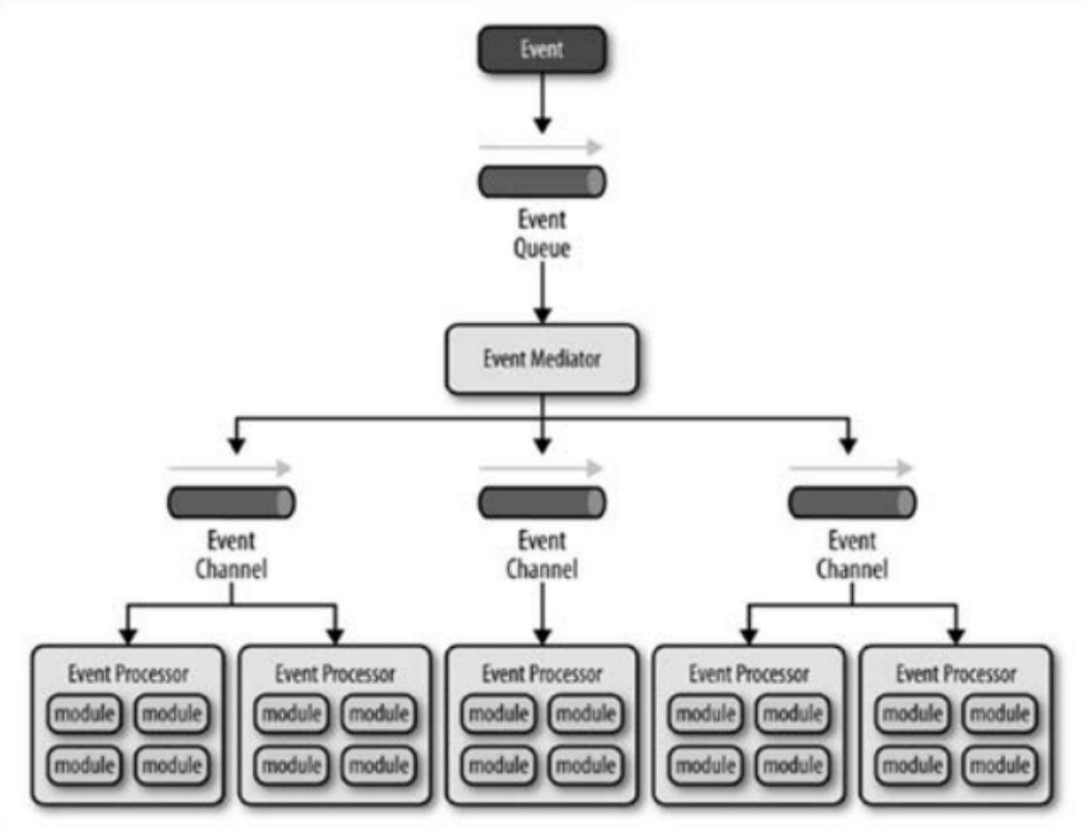


Figura 21: Diagramma del Desing Pattern Event-driven

- **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

10.1.2 MVC

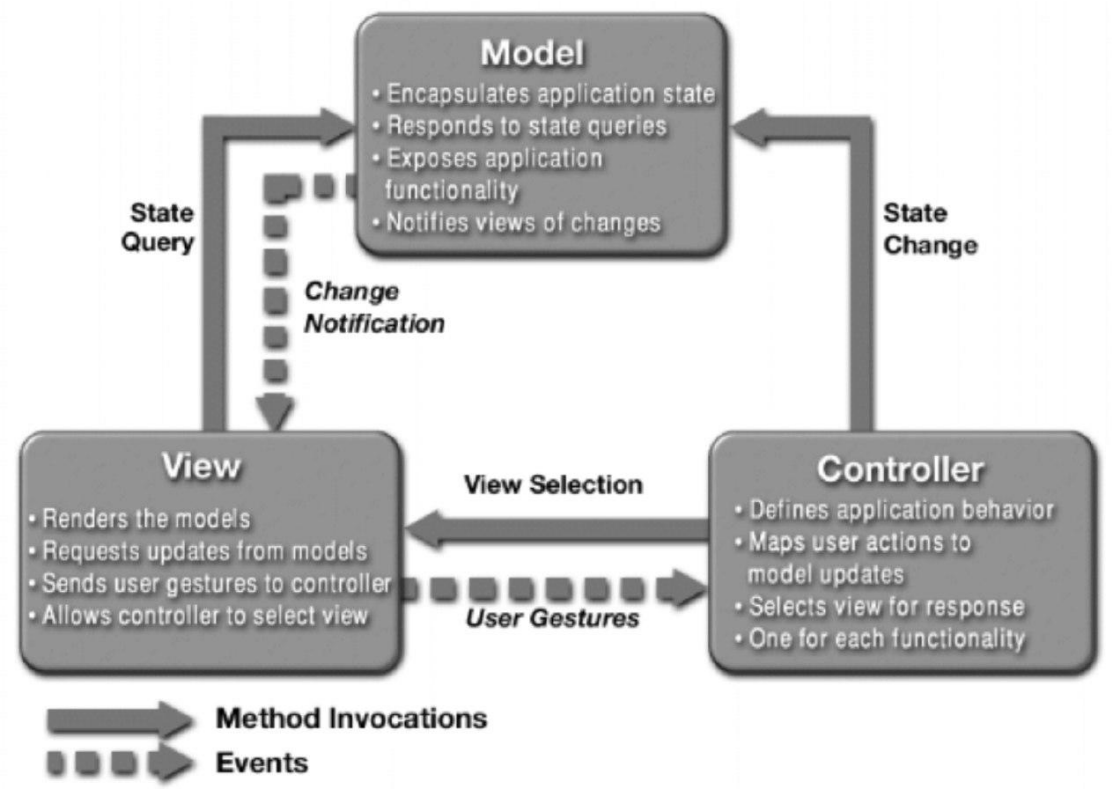


Figura 22: Diagramma del Desing Pattern MVC

- **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

10.1.3 Command

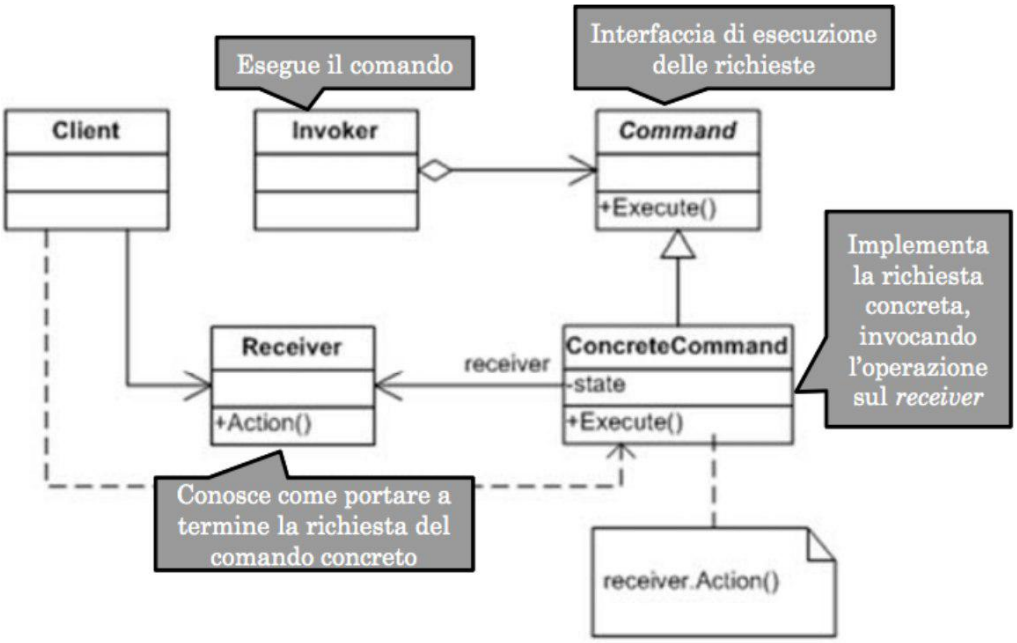


Figura 23: Diagramma del Desing Pattern Command

- **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