pcd-actors

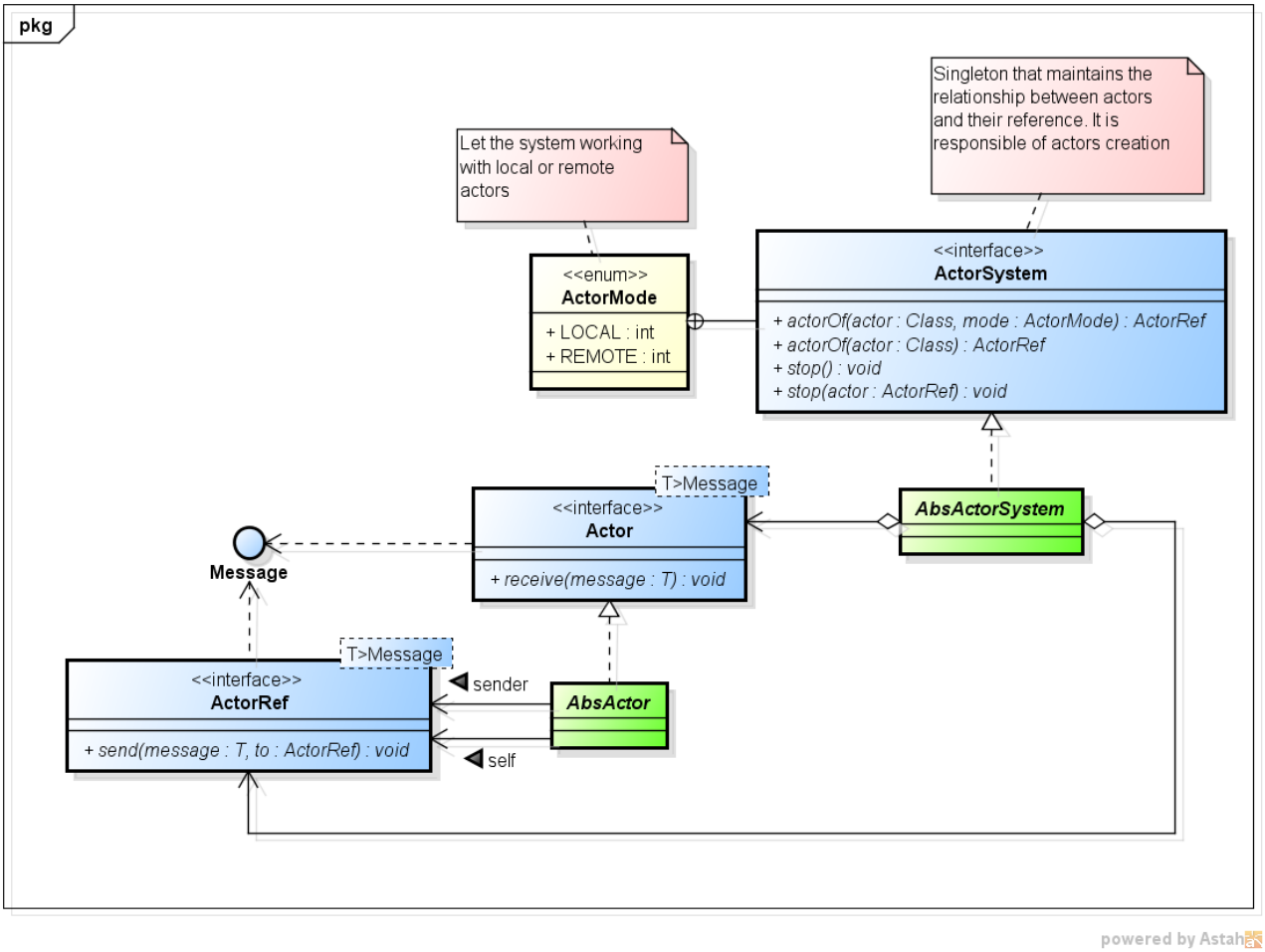
Un sistema che astrae un'implementazione semplificata del modello ad attori. Il sistema deve essere considerato un mock perché i componenti principali sono intenzionalmente lasciati astratti.

I principali tipi astratti del sistema sono i seguenti:

- Actor: Questo tipo rappresenta un attore, il quale può ricevere un messaggio e reagire di conseguenza
- Message: Il messaggio che gli attori si scambiano tra loro. Un messaggio deve contenere un riferimento all'attore che l'ha inviato
- ActorRef: Un riferimento ad un'istanza di un attore. Usando questa astrazione è possibile trattare nello stesso modo attori locali e attori che vengono eseguiti remotamente.
- ActorSystem: Un ActorSystem fornisce le utilità per creare nuove istanze di attori e per individuarli

Architettura Logica

Tutti assieme formano l'architettura software della figura sottostante:



In blu sono colorate le interfacce del sistema, per permettere al sistema di lavorare adeguatamente, ogni interfaccia DEVE avere almeno un'implementazione concreta. In verde sono colorati i tipi che devono essere implementati / estesi / completati.

Segue una breve descrizione di ciascuno dei principali tipi logici di pcd-actors.

Actor

Un attore che appartiene al tipo Actor tiene l'interfaccia dell'attore. L'interfaccia di un attore è identificata dai messaggi ai quali può rispondere. L'interfaccia dell'attore è completamente definita dal metodo:

void receive(T message)

I messaggi ricevuti da un attore non sono immediatamente processati. Devono per forza essere inseriti in una coda dedicata, chiamata mailbox. I messaggi dentro la mailbox devono essere processati in modo asincrono, il che significa che il processare un messaggio non deve bloccare il loop di ricezione degli altri messaggi dell'attore.

L'implementazione dell'attore deve ottimizzare l'uso di thread sincronizzati per soddisfare i suddetti requisiti.

Un Actor ha un riferimento ad attore (vedi il tipo ActorRef) a se stesso e uno all'attore del messaggio correntemente processato.

Messaggi sconosciuti

Nella semplice implementazione richiesta da pcd-actors, se un attore non sa come rispondere ad un particolare tipo di messaggio, viene lanciata un UnsupportMessageException. Questo non è il comportamento standard di un modello ad attori. In una completa implementazione di un modello ad attori, deve essere responsabilità dell'utente decidere quale azione prendere rispetto ad un messaggio sconosciuto.

Inoltre, la politica che ci permette di lanciare un'eccezione in risposta ad un messaggio sconosciuto è possibile perché in pcd-actors un attore non può cambiare la sua interfaccia nel tempo. In realtà, lanciando un'eccezione verrà fermato l'attore, rendendo inutile ogni possibile cambio di interfaccia.

ActorRef

Un riferimento ad un attore (formalmente un ActorRef) è un astrazione del modello usato per indirizzare attori. Ci sono due modalità differenti per indirizzare attori:

- Modalità Locale: L'attore sta girando sulla macchina locale.
- Modalità remota: L'attore potrebbe star girando su una macchina remota.

Usando questa astrazione, un attore remoto può essere usato come un attore locale semplificando il modello di elaborazione.

Una volta che un'istanza di ActorRef è stata ottenuta, è possibile inviare messaggi all'attore corrispondente usando il seguente metodo:

void send(T message, ActorRef to);

Per fare la magia, è necessario usare l'istanza di ActorSystem descritta sotto. I messaggi possono essere inviati solo tra attori. Nessun altro tipo può inviare un messaggio ad un attore.

Message

Un messaggio è un pezzo di informazione che gli attori si inviano tra loro. Ogni messaggio deve essere diviso logicamente in tre parti:

- Un tag, che rappresenta l'operazione richiesta dal messaggio
- Un target, che rappresenta l'indirizzo dell'attore che riceverà il messaggio (il destinatario)
- Il payload, che può rappresentare i dati che devono essere inviati assieme al messaggio

Actor system

Il sistema ad attori (ActorSystem) ha la responsabilità di mantenere i riferimenti di ogni attore creato. Usare l'ActorSystem deve essere l'unico modo per costruire una nuova istanza di un attore. I metodi di fabbricazione resi disponibili dall'ActorSystem sono:

```
ActorRef<? extends Message> actorOf(Class<Actor<?>> actor);
ActorRef<? extends Message> actorOf(Class<Actor<?>> actor, ActorMode mode);
```

Il primo permette di costruire un'istanza locale di un attore del tipo dato. Il secondo permette di decidere se deve essere costruita un'istanza locale o remota.

Il sistema ad attori mantiene la relazione tra ogni attore e il suo riferimento, usando una mappa. La mappa è indicizzata da ActorRef ed è locata dentro il tipo AbsActorSystem. Gli accessi alla mappa devono essere propriamente sincronizzati.

Il sistema ad attori ha anche la responsabilità di fermare un attore e di bloccare l'intero sistema, usando i seguenti metodi:

```
void stop();
void stop(ActorRef<?> actor);
```

Il Sistema ad attori visto come Singleton

Il sistema ad attori DEVE avere una singola istanza attiva. Questa istanza deve essere necessariamente inizializzata nel metodo main del programma.

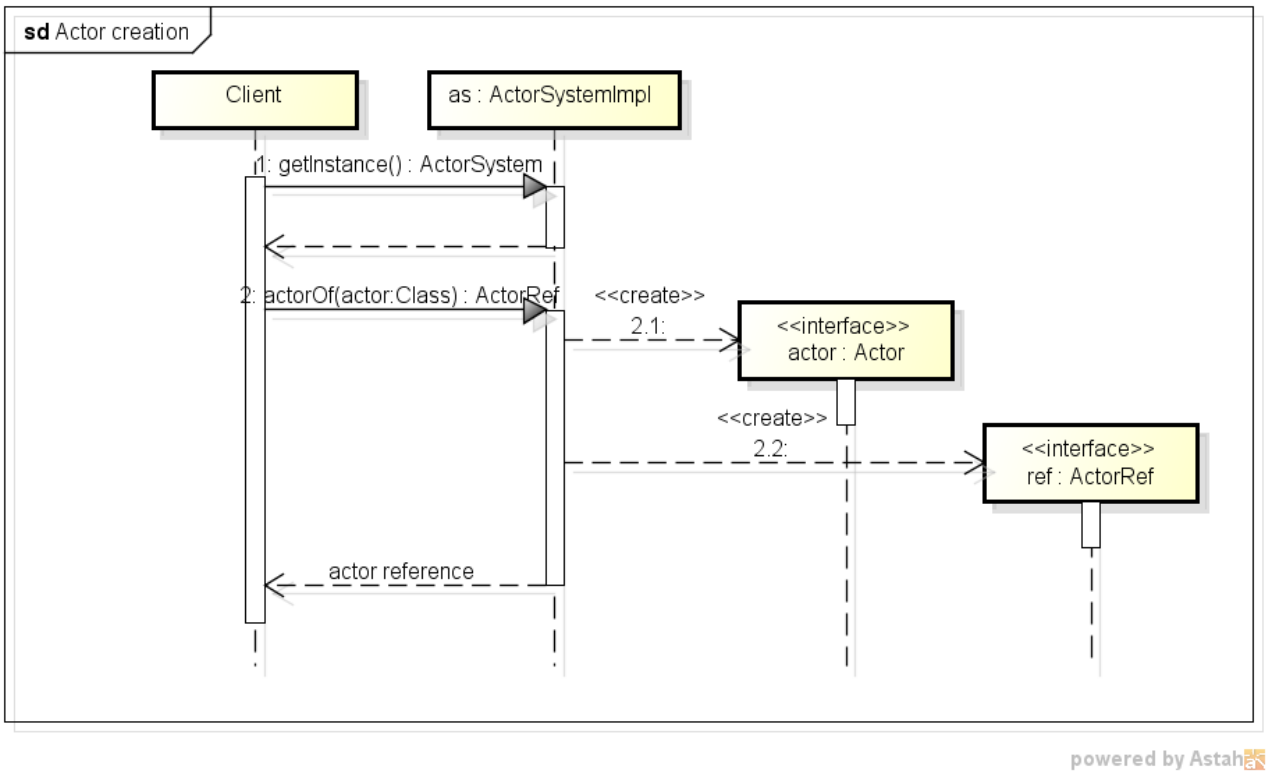
Per implementare correttamente il sistema remoto, questa istanza deve essere serializzabile. Il modo migliore per realizzare questa funzionalità è di usare un *dependence injection* framework , come Google Guice, Spring o CDI. Comunque l'uso di un DI framework va ben al di là degli scopi di questo piccolo progetto.

Quindi, la proprietà precedente deve essere soddisfatta usando altre tecniche, che non usano esplicitamente alcuna forma del pattern Singleton.

Interazioni tra tipi

Questa sezione illustra come i tipi suddetti interagiscono tra loro per realizzare le relative funzionalità.

Creazione di attori

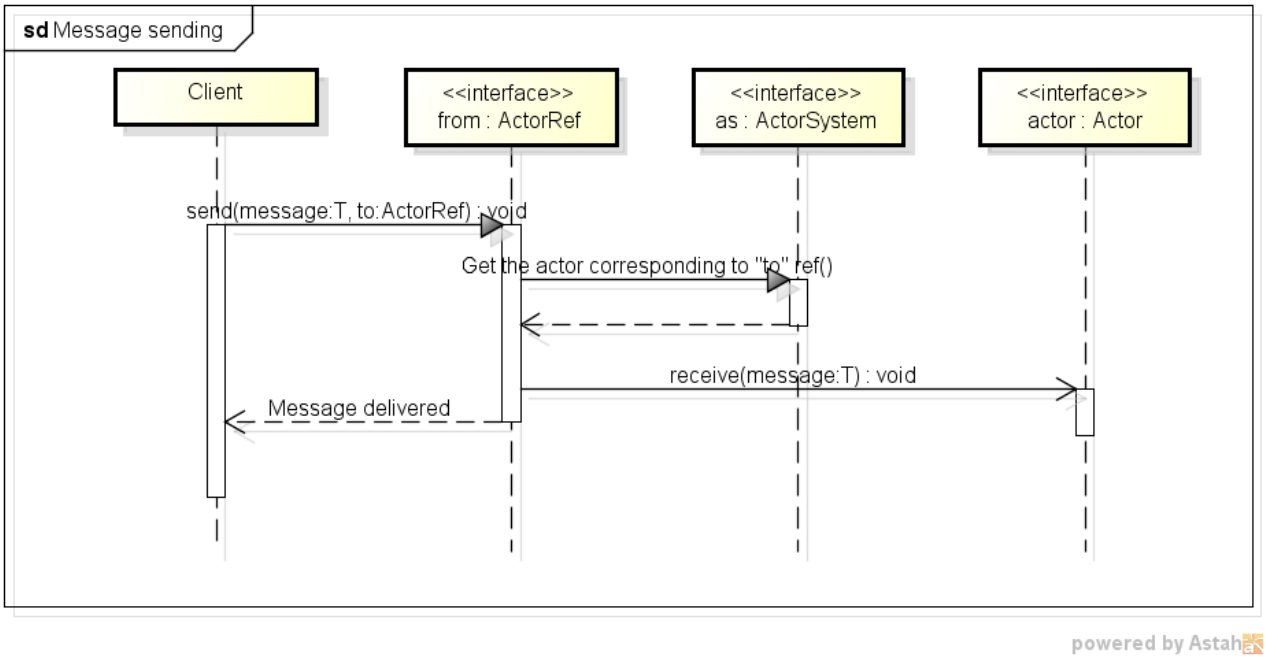


Per creare un nuovo attore, basta chiedere all'ActorSystem di fare il lavoro sporco.

Quindi, prima di tutto, un client deve ottenere un riferimento all'ActorSystem. Usando questo riferimento, chiede al sistema di creare una nuova istanza di un attore. Il risultato di questa richiesta è il riferimento ad attore dell'attore creato.

Invio di Messaggi

Una volta che un client ha ottenuto i riferimenti a due attori, può chiedere al primo di inviare un messaggio al secondo. Chiaramente, per ottenere la reale istanza di un attore (e non il suo riferimento) deve essere interrogato l'ActorSystem.



Il più delle volte, il client sarà lui stesso un attore, che chiede al suo riferimento di inviare un messaggio ad un altro attore.

Building

Il progetto pcd-actors è configurato come un progetto Maven. Specificatamente, è stato generato usando il comando:

```
mvn archetype:generate -DarchetypeGroupId=it.unipd.math.pcd.actors -
DarchetypeArtifactId=pcd-actors -DarchetypeVersion=1.0-SNAPSHOT
```

L'albero delle cartelle generato è il seguente:

```
project
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- App.java
    `-- test
        `-- java
            `-- AppTest.java
```

Come al solito, mettete I file sorgenti sotto la cartella chiamata scr/main/java; Mettete i file di test (di unità e integazione) sotto la cartella scr/test/java.

Per costruire la libreria del sistema ad attori usare il seguente comando:

mvn package

La libreria verrà creata da Maven dentro la cartella target, con il nome pcd-actors.jar.

Per eseguire i test usare il comando

mvn test

L'output della console vi dirà se il processo di build e i test sono stati completati correttamente.

Testing

I Test di ciascuna entità sono fatti con Junit 4. Essendo pcd-actors un progetto Maven, i test sono localizzati nella cartella scr/test/java.

Test integrativi verranno aggiunti nelle prossime settimane. Questi test hanno lo scopo di verificare che l'intero sistema soddisfi i requisiti sopra esposti.

Siete liberi (il che significa che ci si aspetta) di aggiungere i vostri test alla vostra implementazione del sistema ad attori.

FAQ

Il meta framework contiene quattro interfacce e due classi astratte. Cosa devo implementare?
Pensando al processo di testing, sicuramente capirete che i test non possono essere eseguiti su tipi che non esistono nel processo originale. Quindi, non è prevista l'implementazione dell'interfaccia Message, ne di dare un'implementazione concreta di AbsActor.

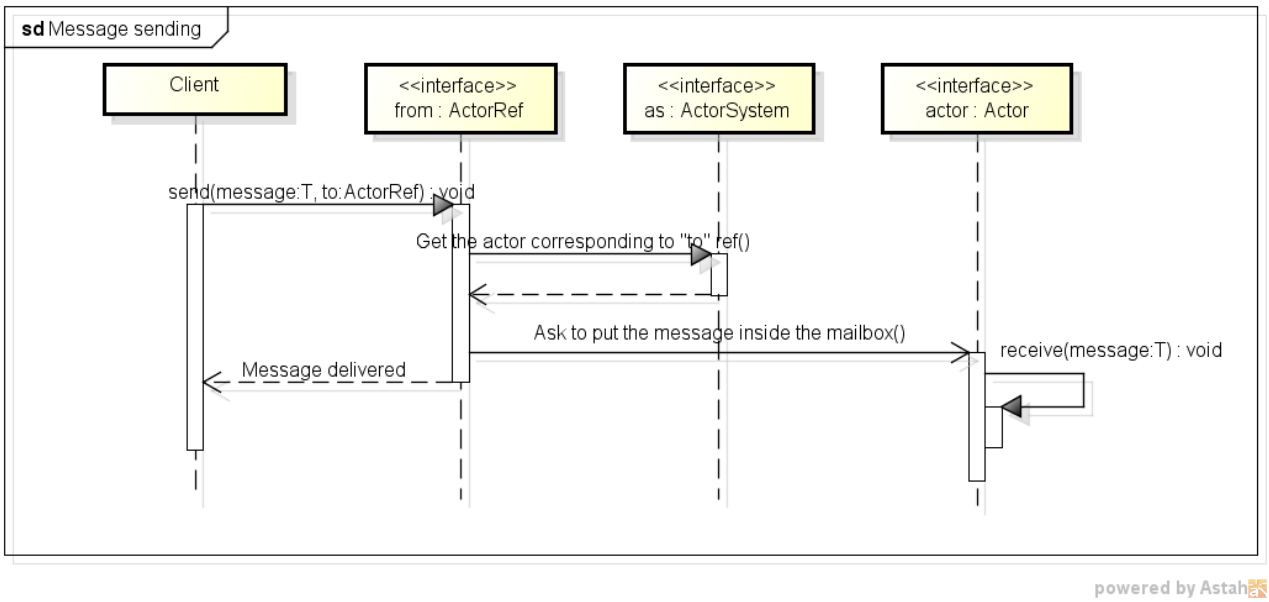
Le implementazioni devono creare tipi concreti o astratti?
Dipende: un utente di pcd-actors non deve dare un'implementazione di ActorSystem, quindi sta a voi fornire una piena implementazione di questo tipo. Anche ActorRef necessità di almeno due implementazioni: Una per gli attori locali, l'altra riferita agli attori remoti.

Dobbiamo implementare alcune strutture dati sincronizzate, Actor e ActorSystem devono essere implementate come thread?
No, non devono. Questi due tipi probabilmente genereranno vari thread per raggiungere i loro scopi, ma non sono intesi come thread loro stessi.

ActorSystem è un Singleton?
In un mondo perfetto e idilliaco useremmo un dependency injection framework, per garantire che ActorSystem sia istanziato una volta sola. Sfortunatamente, non possiamo usare questi framework, a causa della loro complessità. Non potete neanche implementare ActorSystem come Singleton. Allora, dovete essere sicuri che ogni oggetto che deve usare un'istanza di ActorSystem usi la stessa.

Un utente di pcd-actors può estendere ActorSystem?
No, sicuramente non potrà.

Come interagirà ActorRef con il resto del sistema? Qual è il suo ruolo?
ActorRef disaccoppia l'implementazione di un attore dal modo in cui viene invocato da un altro attore. Quindi, ci saranno almeno due implementazioni di questo tipo: una si riferirà agli attori locali, e un'altra a quelli remoti. La sua interazione col resto del sistema è riassunta nel seguente diagramma di sequenza:



La vista logica che è stata fornita per il tipo Message equivale alla sua vista fisica (cioè alla sua implentazione), giusto?
No, la vista logica è stata fornita per descrivere il modello ad attori in generale. In pcd-actors i messaggi sono completamente liberi dall'implementazione, dal punto di vista della libreria.

Come fa un'istanza di Actor ad acquisire il riferimento all'ActorRef del mittente?
Non dovete usare Message per implementare questa funzionalità.

Un utente di pcd-actors può usare solo metodi delle interfacce che sono definiti nel meta-framework, giusto?
Sicuramente si, potete aggiungere qualsiasi metodo vogliate, ma non saranno ne testati ne valutati.

Perché possiamo inviare solo messaggi di tipo Message agli attori?
E' una semplificazione. In questo modo, possiamo definire un protocollo di comunicazione personalizzato tra attori che è sempre lo stesso in tutte le implementazioni della libreria e si basa sui principi orientati agli oggetti.

Se il meta-framework fornisce solo la classe astratta AbsActorSystem, come cavolo utilizzeranno la libereria i test?
Questa è una buona domanda. Penso userò alcune reflection per scoprire quali delle vostre classi estendono AbsActorSystem.

Dobbiamo implementare il metodo receive del tipo Actor per il progetto?
No, non dovete. Questo è l'aggancio che un utente di pcd-actors deve implementare per usare la libreria.

Un client che intende usare entrambi gli attori, sia locali che remoti deve interagire con loro allo stesso modo, no?
Gli attori remoti e locali devono esporre la stessa interfaccia al client.

Come posso capire quanto l'attore deve processare il prossimo messaggio?
Questo è il punto cruciale del progetto e non posso dirvi come implementare questa funzionalità ;)

Un Message può contenere della logica o è un semplice segnaposto che dice ad un attore cosa deve fare?
Come detto prima, un Message non deve avere alcuna implementazione. Riferitevi a questa discussione su Stackoverflow per capire perché: Akka/Java: Handling multiple message types inside a custom actor?

Un attore è un oggetto immutabile? Può un attore cambiare la sua interfaccia?
Un attore può cambiare la sua interfaccia nel modello ad attori originale. In pcd-actors operiamo una semplificazione e non permettiamo ad un attore di cambiare la sua interfaccia nel tempo. Un attore ha uno stato interno che cambia durante il tempo. Quindi non può essere considerato come un oggetto immutabile.

Un attore che invia un messaggio ad un altro attore deve ricevere come ritorno un messaggio di ack/nack, vero?
No. Non vogliamo implementare nessun protocollo di comunicazione predefinito tra attori. Sarà l'utente finale che definirà un protocollo come questo.

Possiamo usare dei framework esterni per implementare il meta-framework, come Spring?
Preferisco che non aggiungiate nessun framework addizionale al progetto originario. L'unico framework che potete usare durante il processo dei test di unità è Mockito.

Possiamo usare strumenti esterni per controllare lo stile del codice che produrremmo?
Si, potete (e dovreste).

Possiamo aggiungere altri metodi alle classi date?
Certo. Potete aggiungere qualsiasi metodo che pensate possa servire. Ma state attenti a non modificare l'interfaccia pubblica dei tipi perché i test d'unità non possono fare affidamento sulla vostra interfaccia personalizzata.

Un ActorRef può chiamare direttamente il metodo receive del corrispondente Actor?
Usando l'architettura attuale, non è possibile. Innanzitutto un *Message* deve essere inserito dentro la mailbox dell'*Actor,* poi il *Message* diventa idoneo per l'elaborazione.

Licenza