**Assignment-03**

**1)Actor Programming**

**Progettazione degli attori**

In questo elaborato sono stati implementati i seguenti attori, il cui compito è descritto di seguito:

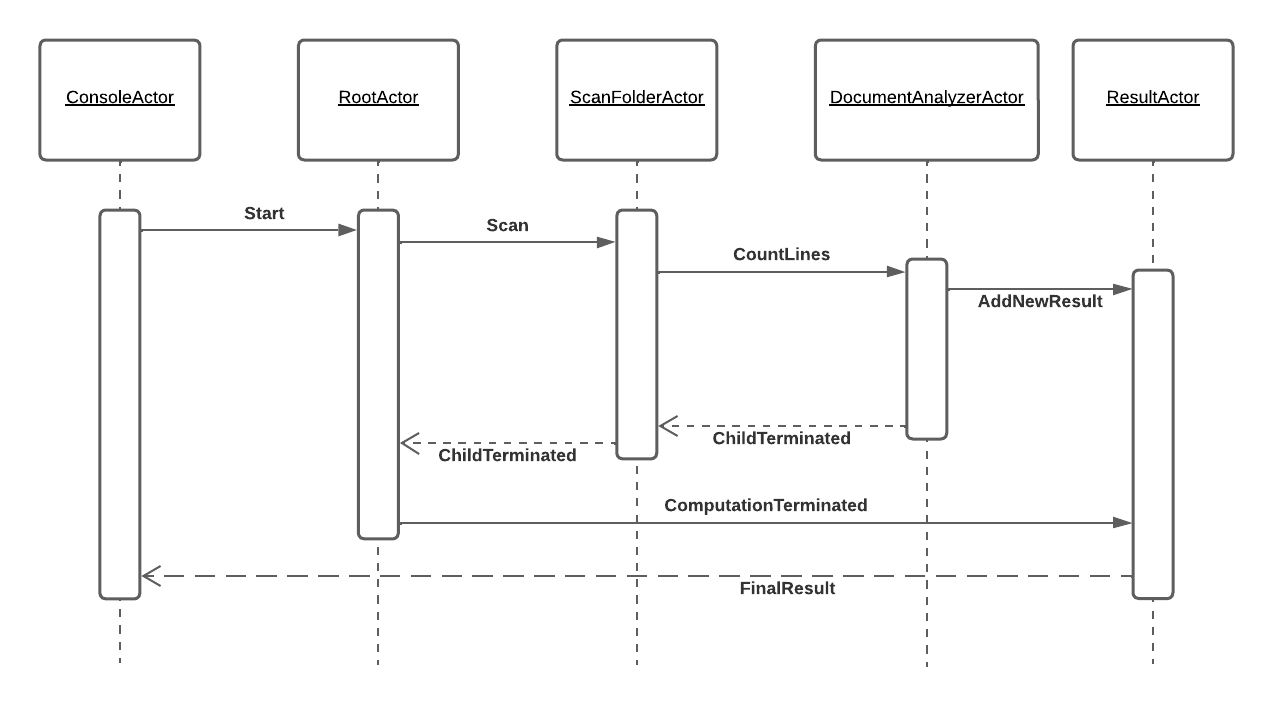
* **RootActor**: Attore che svolge il ruolo di coordinatore e viene creato direttamente dall’ActorSystem, ha il compito di creare a sua volta tre attori principali, il **ConsoleActor/GUIActor**, il **ResultActor** e il primo **ScanFolderActor**.
* **ScanFolderActor**: si occupa dello scanning di una directory, effettuando lo *spawn* di un nuovo **ScanFolderActor** per ogni subfolder e di un nuovo **DocumentAnalyzerActor** per ogni documento.
* **DocumentAnalyzerActor**: ha il compito di contare le righe del documento che gli viene assegnato e di inviare il risultato al **ResultActor**.
* **ResultActor**: riceve i risultati elaborati dal **DocumentAnalyzerActor** e aggiorna periodicamente il ranking e la distribuzione, inviando successivamente il report da mostrare in output al **ConsoleActor/GUIActor**.
* **ConsoleActor**: si occupa di recepire i parametri di input, dare inizio all’esecuzione comunicando con il **RootActor,** ricevere e visualizzare il report finale quando disponibile.
* **GUIActor**: svolge il medesimo compito incapsulato nel **ConsoleActor** ma con la possibilità di fermare l’esecuzione e visualizzare incrementalmente i risultati parziali.

**Scambio di messaggi**

Di seguito viene sintetizzata l’organizzazione dello scambio di messaggi tra gli attori, indicando per ognuno a quali tipologie di comandi reagisce.

* **RootActorCommand**
  + **Start:** comando per avviare l’esecuzione che contiene all’interno le informazioni necessarie per il setup (directory iniziale, numero di files da visualizzare ecc.)
  + **Stop:** Utilizzato per fermare l’esecuzione (viene inviato dal **GUIActor** alla pressione del pulsante Stop).
* **ScanFolderCommand**
  + **Scan:** utilizzato per avviare una scansione, contiene all’interno la directory su cui effettuarla
  + **ChildTerminated:** viene ricevuto ogni volta che un attore figlio termina l’esecuzione e di conseguenza inviato al padre.
* **DocumentAnalyzerCommand**
  + **CountLines:** avvia il conteggio delle righe del file contenuto al suo interno.
* **ResultCommand**
  + **Subscribe:** contiene il riferimento di un nuovo attore che si sottoscrive (utilizzato per avere il riferimento alle view per l’aggiornamento dell’output)
  + **AddNewResult:** ricevuto ogni volta che viene computato un nuovo risultato (che è di conseguenza contenuto al suo interno.
  + **ComputationTerminated:** ricevuto quando l’intera computazione dei files è terminata
* **ViewActorCommand**
  + **IncrementalResult:** contiene al suo interno il risultato incrementale da visualizzare in output.
  + **FinalResult:** contiene il risultato finale da mostrare in output.

**Scenario di esecuzione**



Nello schema di interazione, viene considerato uno scenario di esecuzione in cui (per semplicità di visualizzazione) si considera un progetto con all’interno una directory e un solo file, eseguito in modalità Console.

Sono rappresentati tutti gli attori coinvolti che si scambiano i messaggi indicati nel paragrafo precedente. In un progetto più complicato, il primo **ScanFolderActor** si occupa della creazione di un altro **ScanFolderActor** per ogni subfolder e un **DocumentAnalyzerActor** per ogni documento. Inoltre se consideriamo la modalità GUI, l’unica differenza è l’invio di un messaggio di tipo **IncrementalResult** da parte del **ResultActor**  verso il **GUIActor** ogni volta che è presente un nuovo risultato.