

Actors: Green

Violet: Messages

Główny aktor startowany skryptem z poziomu shell'a. Nie powinien wymagać konta root.

ServedCore

WebCore

Główny aktor startowany z Web aplikacją - tu trzeba jakiegoś stub'a zrobić, żeby np z konsoli przyjmował polecenia, które potem lecą do pozostałych aktorów. Załóż, że WebCore to taki sterownik dla usera. Wszystkie polecenia user'a wychodzą z tego aktora.

Aktor wykonujący polecenia jako konkretny użytkownik.

Commander

Wymagane metody:

df, du, cp, mv, unzip, gunzip, unbz2, untar, rm

Założenie:
Informacje mają być pobierane bez odpalania jakichkolwiek zewnętrznych procesów.

Deployer

Aktor odpowiedzialny za deploy aplikacji użytkownika. (tu tylko stub). Ten kod już jest, biorę to na siebie - dmilith

Założenia i wymagania dodatkowe:

Język implementacji: Scala 2.8

Język preferowany kodu: Angielski, (Polski)

Język preferowany dokumentacji: Angielski, (Polski)

Repozytorium: /git/ServeD.git na draCore.

Dokumentacja: Ma być przynajmniej jedno zdanie opisu co dana klasa/trait/object/metoda robi.

Testowanie: Wymagane spece najważniejszych funkcjonalności.

Poziom anotacji w komentarzach:

HACK - Hacked. Should be done better

FIXME - Bad implementation (but it works!)

XXX - Unclean code

TODO - Todo issue

NOTE - Information note

Opis zadania (w razie czego - pytać dmilith'a):

Wiadomości to case class'y z parametrem/ami. Walidacja poprawności danych/komendy/parametrów wiadomości między aktorami odbywać się ma na poziomie stworzenia wiadomości [np JakiśAktor !! DwieNiepusteWiadomości("Dupa", "")] wywalić ma exceptiona (z wiadomością zwrotną do usera!) jeszcze przed wysłaniem takiej wiadomości do aktora docelowego.

Komunikacja między aktorami ma być zaprojektowana w taki sposób, by umożliwić synchroniczną lub asynchroniczną komunikację (np w zależności od potrzeb, choć raczej będziemy czekać na rezultat).

(np: Commander dostaje synchroniczną wiadomość: CheckFreeSpace("username") od aktora Deployer i zwraca odpowiedź FreeSpaceResult("username") do Commander'a)

keywords:

scala, remote actors (czyli komunikacja lokalnie po tcp, bo to będzie komunikacja między 2ma procesami na 2ch różnych jvm'ach), case class, named params, default args, specializations, traits, objects.

=====

This project is a closed code project

=====

© 2010 by verknowsys.com

® All Rights Reserved