

Programmation parallèle avec des Acteurs

INF7845 - Principes avancés des langages à objets

Antoine Laurent

24 avril 2017

Université du Québec à Montréal

1. Introduction
2. Problématique
3. Modèle acteur théorique
4. Bibliothèque Akka
5. Bibliothèque acteur de Nit
6. Avantages et Inconvénients
7. Programmes réalisés
8. Conclusion

Introduction

Le modèle acteur est un modèle de programmation utilisé surtout pour de la concurrence.

On le retrouve dans :

- Facebook
- Twitter
- Halo 4 : Orléans
- Erlang, Elixir, etc.

Problématique

Modèle Acteur théorique

- Comprendre comment fonctionne le modèle Acteur et ses caractéristiques.

Bibliothèques

- Étudier deux bibliothèques qui implémentent le modèle Acteur.
- Regarder leurs différences.
- Comparer avec le modèle proposé par Hewitt [1].

Implémentation

- Regarder les performances grâce à des programmes fonctionnels.

Modèle acteur théorique

Caractéristiques

- Comme pour la programmation orienté Objet, dans le modèle Acteur tout est objet.
- Un Acteur est persistant.
- Encapsule son état interne.
- Communique via messages.
- Ils sont asynchrones, les messages sont envoyés de manière asynchrones.
- Tolérant aux pannes : «Let it crash».

Actions

- Un Acteur peut envoyer des messages à lui même ou à d'autres Acteurs.
- Prendre des décisions locales à la réception de messages.
- Créer de nouveaux acteurs.

Caractéristiques

- Communiquent par messages.
- Pas de canaux de communications.
- Stratégie du meilleur effort.
- Temps et ordre d'arrivée indéterminés.

Messages

- Adressés aux acteurs par une adresse.
- Les acteurs ont une boîte aux lettres.

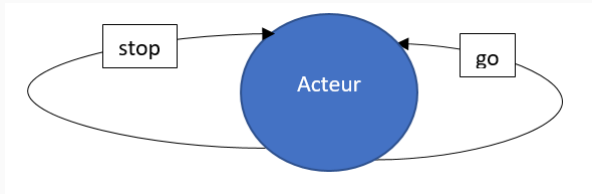


Figure 1 – Exemple de l'indéterminisme du modèle Acteur.

Beaucoup plus simple de faire de la concurrence car :

- Possèdent pas d'état partagé \Rightarrow pas mettre d'états partagés dans les messages.
- Ils procèdent un messages à la fois.

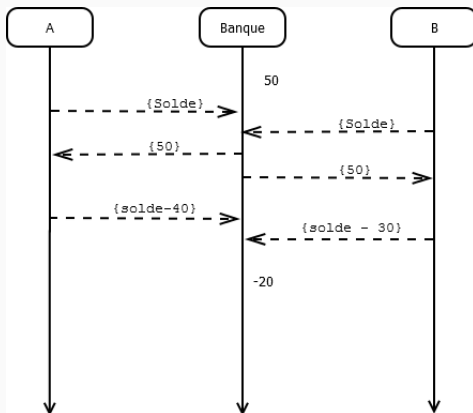
Avantages

- Pas de verrous mortels
- Abstraction du bas niveau
- Mise à l'échelle facile

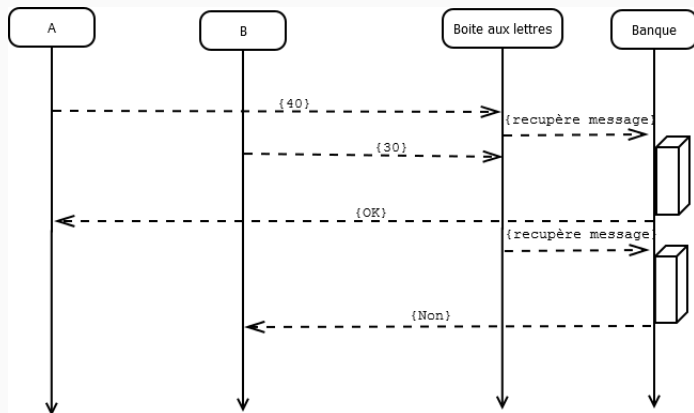
Désavantages

- Moins performant que les threads.
- Canal de communication.

Exemple compte Bancaire



Exemple compte Bancaire



- Threads
- Future
- Petri Nets
- CSP : Communicating sequential processes

- Chaque acteur a un superviseur.
- S'il crash on le laisse faire.
- Le système ne doit pas être perturbé.
- Hiérarchie de superviseurs.

Bibliothèque Akka

Ils sont identifiés par :

- ActorRef : représente l'acteur créé et sert d'adressage lors de l'envoi de messages.
- path : le nom de l'acteur, représenté par sa suite de superviseur.

Example path

- Un path local : akka://SimpleSystem/user/SimpleActor
- Un path à distance :
akka.tcp://my-sys@host.example.com:5678/user/service-b

- On ne peut pas créer un acteur avec `new`.
- Il faut un système d'acteur : `ActorSystem("NomSystème")`.
- Props : une classe de configuration.
- Méthode `actorOf` qui retourne une référence.

```
val system = ActorSystem("mandelbrot")  
val master = system.actorOf(Props(new Master(128,128)),  
    name="master")
```

Un acteur est supervisé par son créateur.

Quatre options pour le superviseur de l'acteur crashé :

- Redémarrer son subordonné.
- Reprendre l'exécution du subordonné.
- Arrêter le subordonné.
- Faire monter l'information.

Un acteur peut :

- Être démarré/redémarré
- Être stoppé : ramassé par le «garbage collector»

Avant chacune de ces action est appelée des méthodes :

- `preStart`
- `preStop`

Tolérance aux pannes - Superviseurs

Trois superviseur de base : Root Guardian, User Guardian et System Guardian.

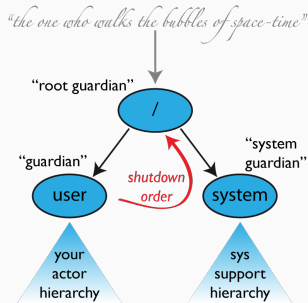


Figure 2 – Trois superviseurs de base [2]

Stratégie par défaut : relancé l'acteur qui a crashé et seulement lui.

Un acteur qui n'est pas superviseur peut monitorer un acteur :

`watch(ActorRef)`.

Il sera informé lors de son arrêt.

Il reçoit le message `Terminate(actorRef)`

Messages

- FIFO.
- Messages pas typés (type Any).
- Pattern matching sur les messages.

```
case object AskNameMessage
class SimpleActor extends Actor {
  import context._
  implicit val timeout = Timeout(FiniteDuration(1, TimeUnit.SECONDS))

  def receive = {
    case AskNameMessage => sender ! "Antoine"
    case _ => context.actorSelection("akka://SimpleSystem/user/SimpleActor2").resolveOne().onComplete {
      case Success(actorRef) => actorRef ! "coucou"
      case Failure(ex) => println("user/" + "somename" + " does not exist")
    }
  }
}
```


Bibliothèque acteur de Nit

Trois entités différentes : Classe, Proxy, Acteur

Acteur a :

- une boîte aux lettres : `shift`, `unsift`, `push` .
- une instance de la classe annoté.
- On spécifie la classe comme acteur avec l'annotation `acteur`.
- On crée le Proxy et l'acteur lors de l'appel de `async`.
- `terminate` et `terminate_now` envoie un message `ShutDownMessage`.



Figure 3 – Schématisation d'un appel asynchrone avec les acteurs dans Nit

Avantages et Inconvénients

Avantages

- Principes fondamentaux du modèle acteur implémenté
- Impression de manipuler un vrai acteur.
- Mise à l'échelle facile avec TCP et HTTP.

Désavantages

- Possibilité d'anti-patterns.
- Pas de typage des Messages.
- Exceptions très complexes.

Avantages

- Messages et état interne.
- Simple à utiliser.
- Pas besoin de changer une classe déjà créée.
- Messages typés.

Désavantages

- Possibilité d'anti-patterns.
- Pas d'adresse pour les acteurs.
- Pas toutes les fonctionnalités du modèle acteur présent.

Programmes réalisés

Mandelbrot

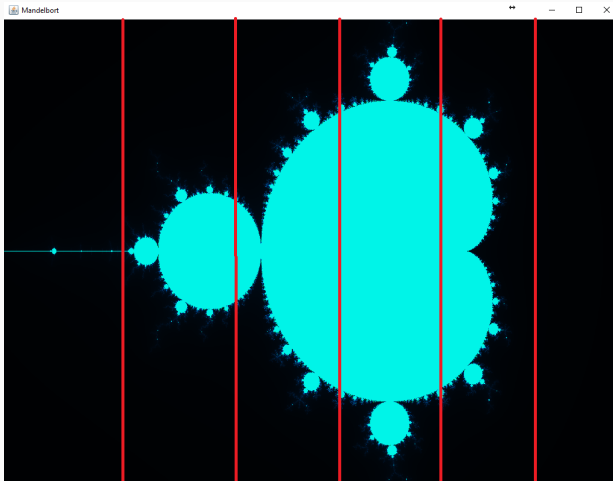


Figure 4 – Découpage de Mandelbrot

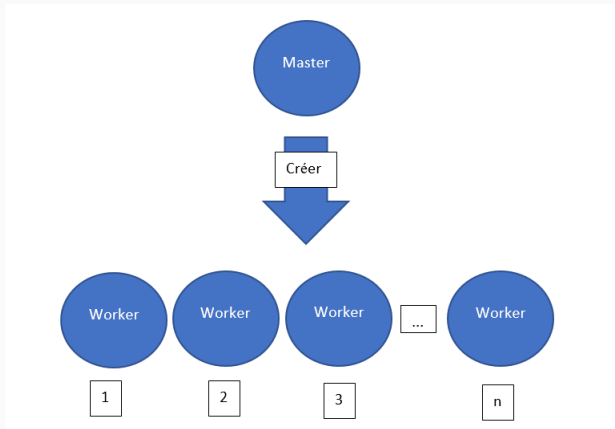


Figure 5 – Représentation du calcul de Mandelbrot avec le modèle Acteur

Scala Actors Vs Java Threads

Nombre de threads/acteurs	Scala	Java
1	65.5 sec	0.7 sec
8	23.3 sec	0.2 sec

Table 1 – Tableau des performances du calcul de Mandelbrot

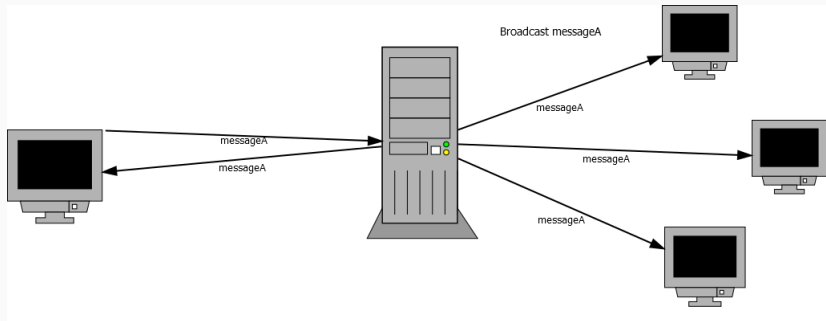


Figure 6 – Représentation du chat avec le modèle Acteur

Conclusion

Merci pour votre attention !
Questions ?



C. Hewitt, P. Bishop, and R. Steiger.

Session 8 formalisms for artificial intelligence a universal modular actor formalism for artificial intelligence.

In *Advance Papers of the Conference*, volume 3, page 235. Stanford Research Institute, 1973.



L. Inc.

Akka scala documentation.

Technical report, 02 2017.