HEIG VD
HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD

Paradigme de concurrence

En utilisant Elixir

2025-03-27

Guillaume Dunant & Edwin Häffner
HEIG-VD
2025
ICSL

Table des matières

I. Introduction

L'idée derrière le paradigme de concurrence est de pouvoir efficacement gérer plusieurs tâches en parallèle sur une même machine, ou même sur plusieurs machines. L'objectif final est d'optimiser l'utilisation des ressources disponibles, de réduire les temps d'attente et d'accélérer l'exécution des programmes en répartissant les calculs ou les opérations entre plusieurs processus ou threads qui s'exécutent simultanément.

Un exemple simple serait d'avoir une application qui utilise un thread du CPU pour afficher la partie GUI (interface graphique) et un autre thread pour effectuer des calculs en arrière-plan. Cela éviterait de bloquer l'affichage graphique lorsqu'un calcul complexe est demandé, améliorant ainsi la fluidité et la réactivité de l'application.

II. Motivations du choix de la concurrence

La concurrence est devenue essentielle dans le développement de logiciels modernes. Les CPU actuels disposent d'un nombre croissant de cœurs, même dans le marché des processeurs abordables, par exemple, un processeur à moins de 100 francs offre déjà au moins 6 cœurs.

Donc il nous faut pouvoir utiliser ces coeurs maintenant si abondant pour pouvoir faire des applications efficace.

La concurrence est aussi très importante lorsqu'on interagit avec des API web et d'autres machines. La communication n'est pour l'instant pas instantanée et il faut alors avoir un mécanisme d'attente qui ne fait pas arrêter le programme. Les opérations d'entrée/sortie (I/O), comme les requêtes réseau ou les accès disque, sont particulièrement concernées car elles comportent des temps d'attente réel.

III. Paradigme de concurrence

La programmation concurrente est donc une façon de programmer en tenant compte de l'existence de ces threads et processus. Elle implique la conception de programmes où plusieurs séquences d'instructions peuvent s'exécuter simultanément ou en alternance rapide (préemption de plusieurs processus entre eux par exemple).

Ce paradigme introduit des concepts spécifiques comme la synchronisation, les verrous, les sémaphores, les variables atomiques, et les files de messages, qui permettent de coordonner l'exécution des différents processus ou threads et d'éviter les problèmes classiques de concurrence comme les race conditions, les deadlocks ou les famines.

IV. Paradigme fonctionnel

Outre le paradigme de concurrence, il nous semblait important de pour une fois effectuer un projet en entier en utilisant un language fonctionnel. Dans le cadre de notre cours de "Paradigme et Language de Programmation", on a pu se mouiller les mains avec Haskell, mais outre les mini programmes que nous avions pu réaliser, nous n'avions jamais fait de "grand" projet du début à la fin en utilisant un language fonctionnel.

Dans la programmation fonctionnelle, une variable est toujours immuable, il n'y a pas de système de reassignement de valeurs, donc les problèmes liés à la synchronisation disparaissent.

V. Pourquoi Elixir est la solution ?

Elixir tourne sur la machine virtuelle d'Erlang qui est nommée BEAM.

Qu'est ce que c'est BEAM ?

BEAM,

VI. Cahier des charges prévisionnel

VII. Recherches effectuées

Cette rubrique est pour l'instant temporaire, mais nous permet de nous mettre à jour sur ce que nous avons fait lors des jours. A refactor plus tard de façon plus propre

13-03-25

Recherche sur l'initalisation de projets phoenix, notamment en utilisant cettre ressource ci : https://hexdocs.pm/phoenix/up_and_running.html

Deplus on a pu trouver un site qui permet de faire des exercices liés à Elixir ici un lien sur la section de la concurrence : https://elixirschool.com/en/lessons/advanced/otp_concurrency

27-03-25

Début du rapport intermédiaire !

VIII. Bibliographie

https://www.erlang-solutions.com/blog/comparing-elixir-vs-java/ https://medium.com/flatiron-labs/elixir-and-the-beam-how-concurrency-really-works-3cc151cddd61

BEAM

https://en.wikipedia.org/wiki/BEAM_(Erlang_virtual_machine) https://www.erlang.org/blog/a-brief-beam-primer/

IX. Feature

Customized items

Figures are customized but this is settable in the template file. You can of course reference them : Listing 1.

Listing 1 — Code example

```rust
fn main() {
  println!("Hello Typst!");
}
```