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

PCO – Laboratoire 03

Simulation concurrente d'un système hospitalier

CATTIN Nadia

REYMOND Nicolas

Programmation Concurrente

HEIG-VD

5 novembre 2025

Introduction au problème

Ce laboratoire implémente une simulation concurrente d'un écosystème de santé composé d'ambulances, d'hôpitaux, de cliniques, de fournisseurs et d'une assurance. Chaque acteur possède son propre cycle de vie et interagit avec les autres via des éléments (patients ou ressources médicales) et des flux financiers (factures et paiements). L'objectif est de modéliser un fonctionnement réaliste avec un cycle journalier :

- les ambulances transportent des patients vers les hôpitaux ;
- les hôpitaux admettent, réhabilitent et transfèrent des patients vers des cliniques lorsque nécessaire ;
- les cliniques traitent des patients en consommant des ressources achetées auprès des fournisseurs ;
- l'assurance rembourse les prestations facturées ;
- chaque entité paie ses employés et gère ses contraintes (lits disponibles, stocks, trésorerie, etc.).

La simulation est pilotée par une horloge qui synchronise le début et la fin des journées pour tous les participants.

Choix d'implémentation

Nous avons supposé qu'un seul thread pouvait accéder simultanément à une même instance d'une classe. Pour éviter les problèmes de concurrence, chaque attribut partagé est protégé par un mutex dédié. Seuls les attributs manipulés exclusivement dans des méthodes privées, donc à des moments maîtrisés, ne nécessitent pas de verrou. Les accès (lecture ou écriture) aux autres attributs sont systématiquement protégés dans les méthodes publiques, ainsi que dans les méthodes privées susceptibles d'être appelées concurremment à celles-ci.

Tests effectués

Pour tester, nous utilisons la commande `./pco_hospital` qui prend comme arguments le nombre de jours, le nombre de fournisseurs, le nombre d'assurances, le nombre de cliniques, le nombre d'hôpitaux et le nombre d'ambulances :

```
NB_DAYS      = atoi(argv[1]);      DEFAULT = 6;
NB_SUPPLIER  = atoi(argv[2]);      DEFAULT = 3;
NB_INSURANCE = DEFAULT_INSURANCE;  DEFAULT = 1;
NB_CLINICS   = atoi(argv[4]);      DEFAULT = 3;
NB_HOSPITALS = atoi(argv[5]);      DEFAULT = 2;
NB_AMBULANCE = atoi(argv[6]);      DEFAULT = 2;
```

Ici, l'assurance est fixée à 1.

Nous avons effectué plusieurs tests avec différents nombres d'ambulances, d'hôpitaux, de cliniques et de fournisseurs sur une période de 30 jours. Voici quelques exemples de configurations testées :

- Test 1 : valeurs par défaut (6 jours, 3 fournisseurs, 1 assurance, 3 cliniques, 2 hôpitaux, 2 ambulances
- Test 2 : valeurs minimales (1 jour, 1 ambulance, 1 hôpital, 1 clinique, 1 fournisseur)

- Test 3 : grandes valeurs (50 jours, 50 ambulances, 50 hôpitaux, 50 cliniques, 50 fournisseurs)
- Test 4 : très grandes valeurs (500 jours, 500 ambulances, 500 hôpitaux, 500 cliniques, 500 fournisseurs)

Commandes exécutées et leurs résultats :

```
# Test 1 - valeurs par défaut
./pco_hospital

The expected fund is : 4700 and you got at the end : 4700
The expected patient is : 900 and you got at the end : 900

# Test 2 - valeurs minimales
./pco_hospital 1 1 1 1 1 1

The expected fund is : 2700 and you got at the end : 2700
The expected patient is : 900 and you got at the end : 900

# Test 3 - grandes valeurs
./pco_hospital 50 50 1 50 50 50

The expected fund is : 79400 and you got at the end : 79400
The expected patient is : 15300 and you got at the end : 15300

# Test 4 - très grandes valeurs
./pco_hospital 500 500 1 500 500 500

The expected fund is : 784400 and you got at the end : 784400
The expected patient is : 150300 and you got at the end : 150300

# Test 5 - Beaucoup trop grandes valeurs
./pco_hospital 1000 1000 1 1000 1000 1000

The expected fund is : 1567800 and you got at the end : 1567800
The expected patient is : 300600 and you got at the end : 300600
```

Remarque : l'argument `NB_INSURANCE` (3e) est requis par la syntaxe, mais est forcé à 1 par le programme.

Lorsque nous avons testé le programme initialement, nous avons toujours eu la même erreur, avec un nombre de patients attendu de 900, quels que soient les paramètres. Nous avons finalement remarqué qu'il fallait corriger le main, en changeant la ligne 148 en `int startPatient = INITIAL_PATIENT_SICK * ambulances.size();` (ajout du facteur du nombre d'ambulances pour calculer le nombre de patients attendus).

Nous avons également testé d'augmenter uniquement le nombre de jours :

```
for i in (seq 1 5); ./pco_hospital 6000 3 1 3 2 2 2>&1 | grep "expected fund"; end
The expected fund is : 4700 and you got at the end : 4700
The expected fund is : 4700 and you got at the end : 4680
```

```
The expected fund is : 4700 and you got at the end : 4710
The expected fund is : 4700 and you got at the end : 4690
The expected fund is : 4700 and you got at the end : 4680
```

On remarque que le résultat change. Nous avons tenté de trouver où se situe notre problème de concurrence, mais sans succès.

Utilisation de l'IA

Nous avons utilisé plusieurs IA pour ce laboratoire :

- Github Copilot (modèle GPT 5.0 mini) pour nous aider à faire passer le dernier test qui ne passait pas : `Run_EndToEnd_TreatsAndTransfersAtLeastOnce`, ce qui nous a permis de corriger un deadlock que nous n'arrivions pas à comprendre.
- L'IA intégrée d'overleaf, modèle GPT, pour corriger les fautes d'orthographe et de formulation.
- NotebookLM pour relire notre rapport, nous aider à le clarifier et à vérifier qu'il est complet.