Génie Logiciel

1 Février, 2019

Promotion Neumann (5e année)

Projet : Distributeur de boissons

Vincent CANDAPPANE

Amine KAHLAOUI

Hicham CHAALI

Dr. Ouassila LABBANI
NARSIS[2]
Maître de Conférences,
Institut Marey Maison de la Métallurgie (I3M),
64 rue de Sully, Dijon

Table des matières

Table des figures

Introduction

Le génie logiciel est l'ensemble des activités de conception et de mise en œuvre des produits et des procédures tendant à rationaliser la production du logiciel et son suivi. Il s'agit en général d'une activité par laquelle le code source d'un logiciel est spécifié puis produit. Ses techniques de fabrication assurent 3 facteurs majeurs : respect des exigences, respect de la qualité, respect des délais et des coûts.

Ce projet nous permettra en partie la manipulation du langage LOTOS (Language Of Temporal Ordering Specification)[1] qui est un langage de spécification de processus réactifs et concurrents. Nous soulignons qu'un système distribué est représenté par un ensemble de processus [1] communiquant. Le projet sera ici réalisé via l'outil LotoStem [2].

Appliquées au domaine du Génie Logiciel, nous modéliserons à la LOTOS un distributeur de boissons suivant les règles décrit dans la première partie. Au fil de l'avancement, nous verrons une amélioration de l'automatisation qui sera faite afin de répondre à certaines propriétés et notamment certaines vérifications. Nous soulignons aussi que le but du TP ne se résume pas à une et une seule solution, mais plusieurs. Nous mettrons en place une solution qui fonctionne, mais qui peut en effet être optimisé par la suite.

1. Un processus = une boîte noire capable d'interagir avec son environnement (e.g. d'autres processus) et d'effectuer des actions internes

2. Outil de vérification formelle basé sur la sémantique de maximalité. Il contient un compilateur pour Basic LOTOS qui permet de générer un système de transitions étiquetées basé sur la maximalité (STEM), un model-checker qui permet de vérifier des propriétés exprimées en CTL sur le STEM généré, et un module de visualisation graphique du STEM

1. Modélisation d'un distributeur de boissons

1.1 Fonctionnement du distributeur de boissons

On souhaite modéliser un distributeur de boissons. Les caractéristiques de cette machine sont les suivantes :

- La machine accepte des pièces de 0,20€ et de 1,00€ ,
- Un café coûte 1,00€ alors qu'un thé coûte 0,60€ ,
- Si une pièce de 1,00€ est insérée et qu'un thé est sélectionné, la monnaie (0,40€) est rendue,
- Une fois qu'une pièce est insérée, il n'est pas possible d'annuler la transaction.

FIGURE 1.1: Aperçu d'un distributeur de boissons

1.2 Spécification du distributeur de boissons en LOTOS

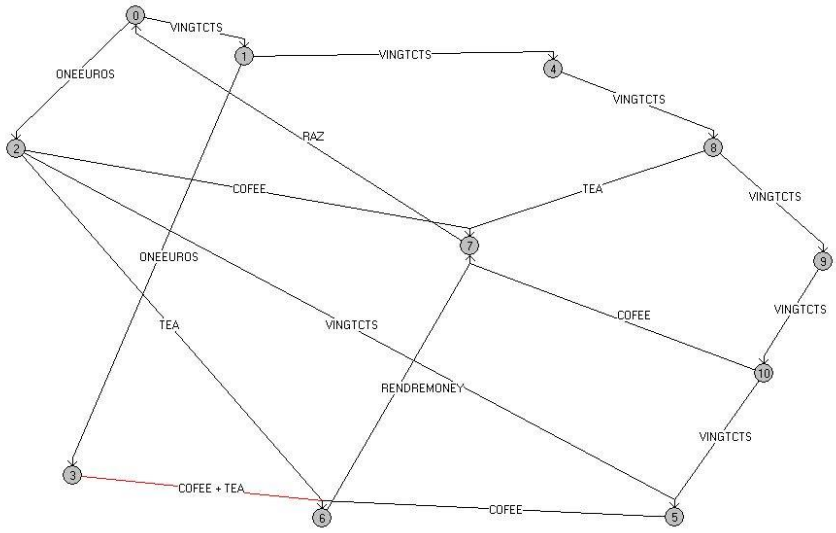

FIGURE 1.2: Modélisation du distributeur de boissons

```
System cofeemachine[vingtCts,oneEuros,cofee,tea,rendremoney,raz,takecofee] :=
                    P1[vingtCts,oneEuros,cofee,tea,rendremoney,raz]

where
Process P1[vingtCts,oneEuros,cofee,tea,rendremoney,raz]:=
vingtCts;(oneEuros;(cofee;rendremoney;raz;P1[vingtCts,oneEuros,cofee,tea,rendremoney,raz]
[]
tea;rendremoney;raz;P1[vingtCts,oneEuros,cofee,tea,rendremoney,raz])
[]
vingtCts;vingtCts;(tea;raz; P1[vingtCts,oneEuros,cofee,tea,rendremoney,raz]
[]
vingtCts;vingtCts;(cofee;raz;P1[vingtCts,oneEuros,cofee,tea,rendremoney,raz]
[]
vingtCts;cofee;rendremoney;raz;P1[vingtCts,oneEuros,cofee,tea,rendremoney,raz])))
[]
oneEuros;(vingtCts;cofee;rendremoney;raz;P1[vingtCts,oneEuros,cofee,tea,rendremoney,raz]
[]
(tea;rendremoney;raz;P1[vingtCts,oneEuros,cofee,tea,rendremoney,raz]
 [] cofee;raz;P1[vingtCts,oneEuros,cofee,tea,rendremoney,raz]) )
endproc

endsys
```

FIGURE 1.3: Spécification du distributeur de boissons en LOTOS

Afin de faciliter la compréhension de chacun des événements entre les différents états, nous les avions nommés de manière à les rendre reconnaissables compte tenu des règles à respecter de l'énoncé. En voici une légende plus ou moins détaillée :

- **ONEEUROS :** Insertion de 1€.
- **VINGTCTS :** Insertion de 0,20€.
- **COFEE :** Choix du "café" (bouton pressé).
- **TEA :** Choix du "thé" (bouton pressé).
- **COFFEE + TEA :** Choix du "Café" ou "Thé" (bouton pressé).
- **RENDREMONEY :** Le distributeur rend la monnaie en plus.
- **RAZ :** Retour à l'état initial 0.

1.3 Spécification d'un client qui désire un café en LOTOS

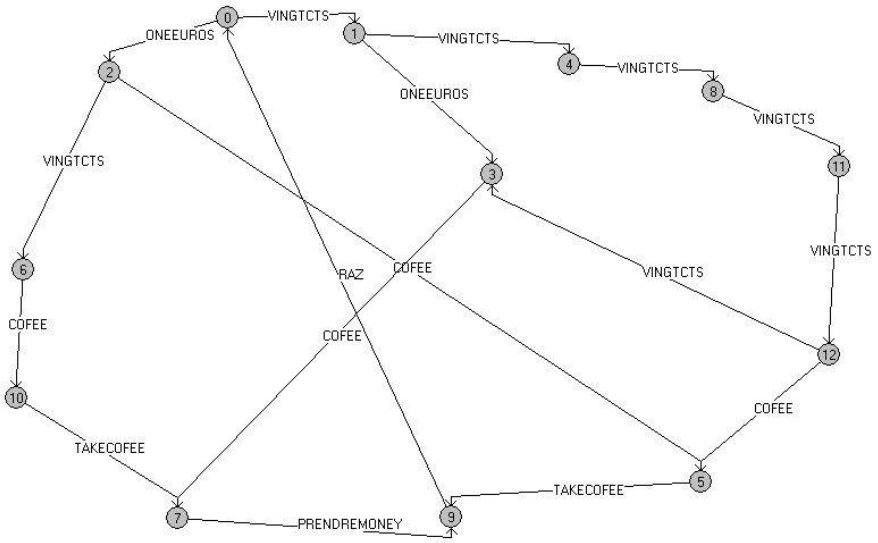

FIGURE 1.4: Modélisation d'un client qui série un café

```
System  client[vingtCts,oneEuros,cofee,takecofee,prendremoney,raz] :=
P2[vingtCts,oneEuros,cofee,takecofee,prendremoney,raz]
where
Process P2[vingtCts,oneEuros,cofee,takecofee,prendremoney,raz]:=

vingtCts;(oneEuros;cofee;prendremoney;raz;P2[vingtCts,oneEuros,cofee,takecofee,prendremoney,raz]
[]

vingtCts;vingtCts;vingtCts;vingtCts;(
vingtCts;cofee;prendremoney;raz;P2[vingtCts,oneEuros,cofee,takecofee,prendremoney,raz]
[]
cofee;takecofee;raz;P2[vingtCts,oneEuros,cofee,takecofee,prendremoney,raz] ) )
[]
oneEuros;(cofee;takecofee;raz;P2[vingtCts,oneEuros,cofee,takecofee,prendremoney,raz]
[]
vingtCts;cofee;takecofee;prendremoney;raz;P2[vingtCts,oneEuros,cofee,takecofee,prendremoney,raz])

endproc
endsys
```

FIGURE 1.5: Spécification d'un client qui désire un café en LOTOS

Afin de faciliter la compréhension de chacun des événements entre les différents états, nous les avions nommés de manière à les rendre reconnaissables compte tenu des règles à respecter de l'énoncé. En voici une légende plus ou moins détaillée :

- **ONEEUROS :** Insertion de 1€.
- **VINGTCTS :** Insertion de 0,20€.
- **COFEE :** Choix du "café" (bouton pressé).
- **TAKECOFEE :** Le client récupère le "café" souhaité.
- **PRENDREMONEY :** Le client récupère la monnaie rendue par le distributeur.
- **RAZ :** Retour à l'état initial 0.

2. Parallélisation de processus

Afin d'avoir un produit fini et fonctionnel, l'on devra ici réaliser une modélisation répondant aux critères du distributeur de boissons, mais aussi du client. Il s'agit ici de mettre en parallèle le processus du distributeur de boissons(P1) et celui du client(P2).

2.1 Modélisation de la parallélisation des processus

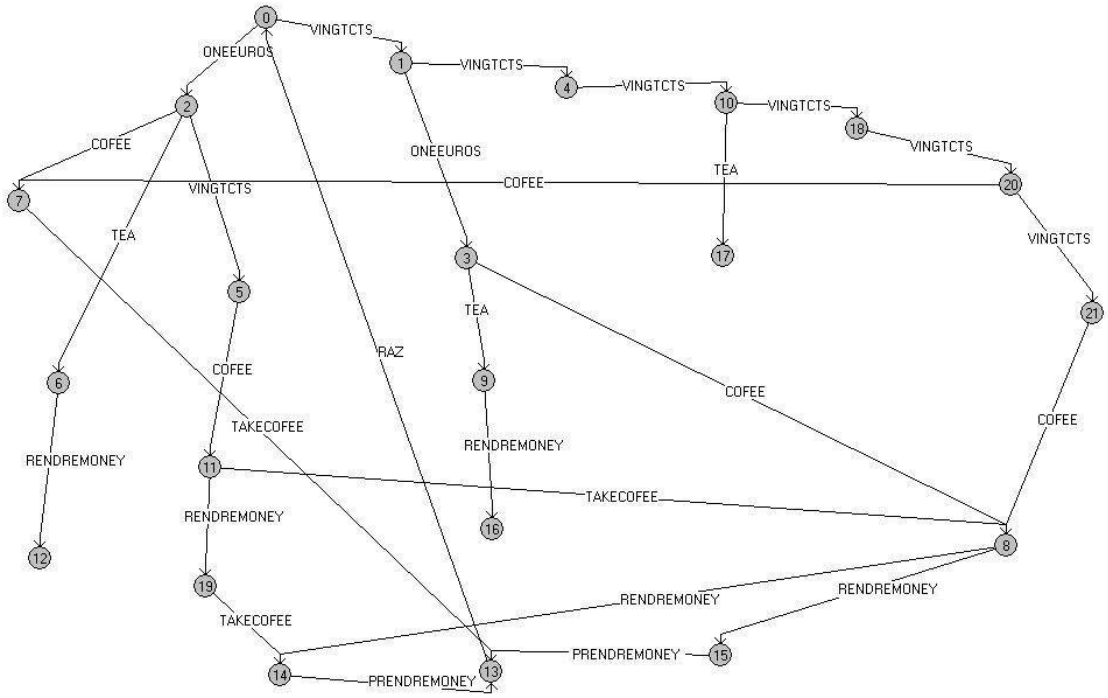

FIGURE 2.1: Modélisation de la parallélisation des 2 processus

À chaque fois qu'on retrouve le même événement dans le P1 et P2, l'on synchronise les 2 processus à cet événement-là. L'on créer donc au final, un processus compacté sans redondance d'événement identique. Pour cela, l'on a fait la synchronisation de P1 et P2 sur les 4 événements suivants : (0,20€ + 1€ + COFEE + RAZ). Mais l'on aurait très bien pu synchroniser sur 3 évènements comme sur 2, cependant cela implique une complexité différente. Nous avions donc fait le choix de synchroniser sur 4 éléments.

2.2 Spécification LOTOS qui met en parallèle les processus P1 et P2

```
System cofeemachine[vingtCts,oneEuros,cofee,tea,rendremoney,prendremoney,takecofee,raz] :=
                    P1[vingtCts,oneEuros,cofee,tea,rendremoney,raz]
                    |[vingtCts,oneEuros,cofee,raz]|
                    P2[vingtCts,oneEuros,cofee,takecofee,prendremoney,raz]

where
Process P1[vingtCts,oneEuros,cofee,tea,rendremoney,raz]:=
vingtCts;(oneEuros;(cofee;rendremoney;raz;P1[vingtCts,oneEuros,cofee,tea,rendremoney,raz]
[]
tea;rendremoney;raz;P1[vingtCts,oneEuros,cofee,tea,rendremoney,raz])
[]
vingtCts;vingtCts;(tea;raz; P1[vingtCts,oneEuros,cofee,tea,rendremoney,raz]
[]
vingtCts;vingtCts;(cofee;raz;P1[vingtCts,oneEuros,cofee,tea,rendremoney,raz]
[]
vingtCts;cofee;rendremoney;raz;P1[vingtCts,oneEuros,cofee,tea,rendremoney,raz])))
[]
oneEuros;(vingtCts;cofee;rendremoney;raz;P1[vingtCts,oneEuros,cofee,tea,rendremoney,raz]
[]
(tea;rendremoney;raz;P1[vingtCts,oneEuros,cofee,tea,rendremoney,raz]
 [] cofee;raz;P1[vingtCts,oneEuros,cofee,tea,rendremoney,raz]) )
endproc
Process P2[vingtCts,oneEuros,cofee,takecofee,prendremoney,raz]:=

vingtCts;(oneEuros;cofee;prendremoney;raz;P2[vingtCts,oneEuros,cofee,takecofee,prendremoney,raz]
[]

vingtCts;vingtCts;vingtCts;vingtCts;(
vingtCts;cofee;prendremoney;raz;P2[vingtCts,oneEuros,cofee,takecofee,prendremoney,raz]
[]
cofee;takecofee;raz;P2[vingtCts,oneEuros,cofee,takecofee,prendremoney,raz] ) )
[]
oneEuros;(cofee;takecofee;raz;P2[vingtCts,oneEuros,cofee,takecofee,prendremoney,raz]
[]
vingtCts;cofee;takecofee;prendremoney;raz;P2[vingtCts,oneEuros,cofee,takecofee,prendremoney,raz])

endproc
endsys
```

FIGURE 2.2: Spécification LOTOS qui met en parallèle les processus P1 et P2

3. Vérification des propriétés

Suivant la complexité du code de départ, les propriétés peuvent être nombreuses. Nous en avons noté 3 qui pourront régler les failles techniques du fonctionnement du distributeur de boissons :

— Si un client demande un café, il finira par l'avoir,
— Il est impossible que le distributeur serve du café et du thé en même temps,
— Il est impossible d'avoir un café pour moins de 1,00€.

3.1 Vérification 1

Condition à régler : Si un client demande un café, il finira par l'avoir.

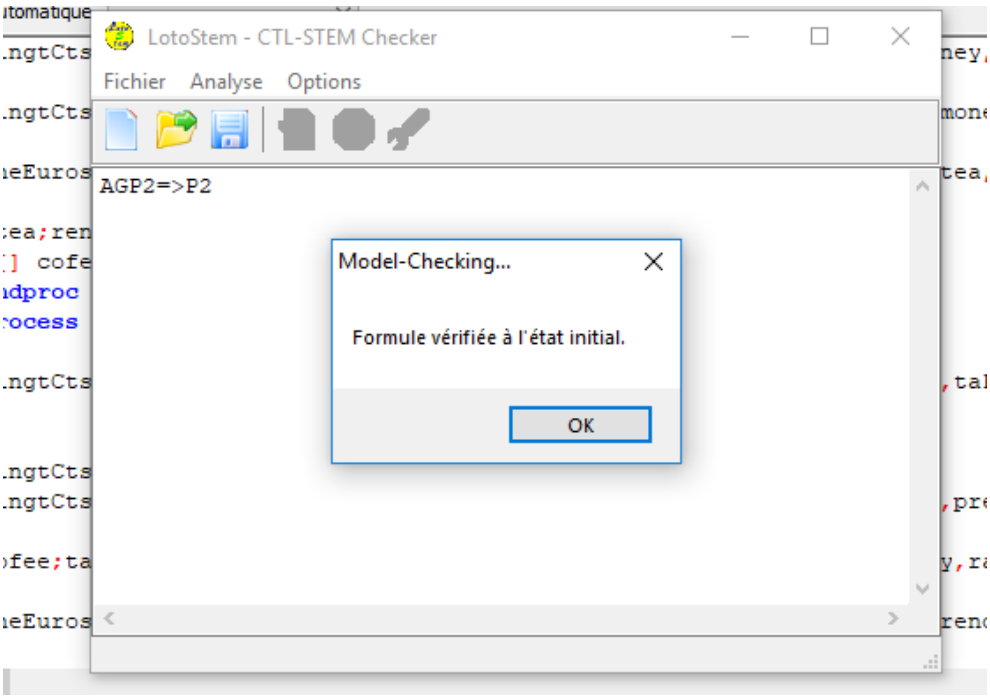

FIGURE 3.1: Vérification 1 = AGP2=>P2

Donc explicitement, quel que soit le chemin/décision que l'on prendra pour prendre un café (5 fois 20 centimes ou une fois 1 euro) on finira toujours par avoir notre café.

3.2 Vérification 2

Condition à régler : Il est impossible que le distributeur serve du café et du thé en même temps.

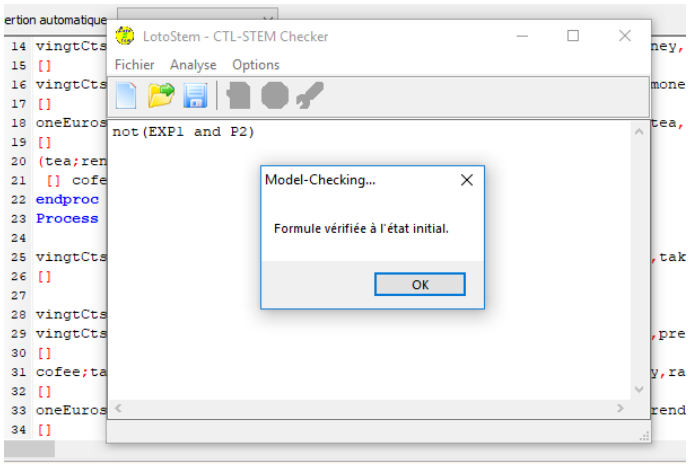

FIGURE 3.2: Vérification 2 = not(EXP1 and P2)

Donc explicitement, le distributeur ne pourra jamais réaliser un café et un thé en même temps, car les processus se suivent dans le temps : c'est soit l'un, soit l'autre.

3.3 Vérification 3

Condition à régler : Il est impossible d'avoir un café pour moins de 1,00€.

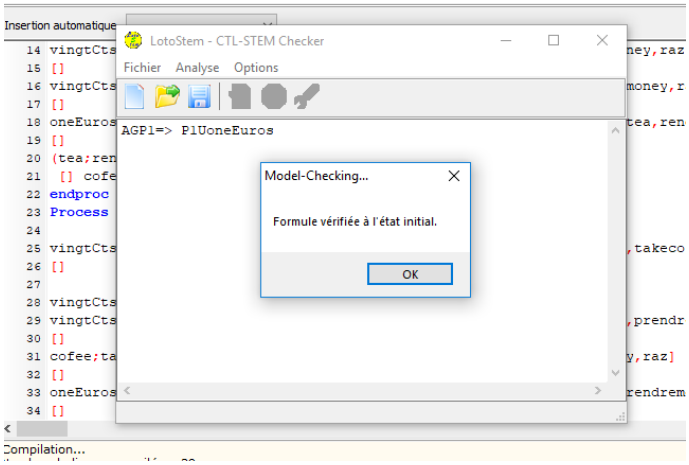

FIGURE 3.3: Vérification 3 : AGP1=> P1UoneEuros

Donc explicitement, le distributeur ne pourra en aucun cas distribuer un café tant qu'on n'aura pas mis une somme minimale de 1€.

Conclusion

Le module du Génie logiciel nous a permis d'avoir une idée concrête du rôle de l'ingénieur au sein d'une entreprise. Le travail d'un ingénieur en informatique est de choisir la bonne approche de Génie logiciel en fonction du projet, car une bonne approche aidera à bien livrer le produit final. Tous en considérant que le produit souhaité et le contexte déterminent la bonne approche.

Le développement d'une application en appliquant les règles du Génie logiciel exige de :

- **Procéder par étapes :** prendre connaissance des besoins, effectuer l'analyse, trouver une solution informatique, réaliser, tester, installer, assurer le suivi.
- **Procéder avec méthode :** du général au détail et au technique, fournir une documentation, s'aider de méthodes appropriées.
- **Savoir se remettre en question :** Est-ce la bonne construction ?, Est-ce le bon produit ?
- **Choisir une bonne équipe :** trouver les compétences, définir les missions de chacun, coordonner les actions.
- **Contrôler les coûts et délais :** aspect économique, bonne maîtrise de la conduite du projet, investissements au bon moment.
- **Garantir le succès du logiciel :** répondre à la demande, assurer la qualité du logiciel.
- **Envisager l'évolution du/de :** logiciel, matériel, l'équipe.

Nous terminerons ce rapport par une loi de l'ingénieur américain Edward A. Murphy Jr.[3] qui résume plus ou moins le module du Génie Logiciel qui est plus communément connu sous le titre de *"Loi de Murphy : Tout ce qui est susceptible de mal tourner tournera mal"*.

Bibliographie

[1] Hubert GARAVEL. *Introduction au langage LOTOS*. URL : ftp://ftp.inrialpes.fr/pub/vasy/publications/cadp/Garavel-90-b.pdf.

[2] Ouassila LABBANI NARSIS. *Maitre de Conférence CIAD*. URL : http://le2i.cnrs.fr/-Ouassila-Labbani-.

[3] Yew-Kwang NG. *A Mathematical Proof and an Application to the Creation of Our Sub-universe*. URL : http://article.sapub.org/10.5923.j.am.20130302.01.html.