Modèle du système d'échéanciers pour la simulation répartie synchrone et asynchrone

Viceić Predrag, viceic@net2000.ch

10 juin 2002

Travail effectué à l'Université de Neuchâtel avec Prof. Jean-Pierre Müller

Table des matières

1 Introduction **5**

2 Simulation événementielle répartie **6**
- 2.1 Modèle d'exécution pessimiste . . . 6
  - 2.1.1 Graphes circulaires dans le modèle d'exécution pessimiste . . . 7
- 2.2 Modèle d'exécution optimiste . . . 8

3 But du projet **10**

4 Étude préliminaire **11**
- 4.1 Bases . . . 11
  - 4.1.1 Définitions . . . 11
- 4.2 Modèle d'exécution synchrone . . . 11
- 4.3 Modèle d'exécution asynchrone . . . 12
- 4.4 Modèle asynchrone pessimiste . . . 12
- 4.5 Modèle asynchrone optimiste . . . 12

5 Propositions de conception des échéanciers **15**
- 5.1 Échéancier optimiste . . . 15
  - 5.1.1 Boucle principale . . . 15
  - 5.1.2 Rollback . . . 15
  - 5.1.3 Gestion du Temps virtuel global (GVT) . . . 18
  - 5.1.4 Gestion de la taille du tampon des rollbacks . . . 19
- 5.2 Échéancier pessimiste . . . 19
- 5.3 Échéancier synchrone . . . 19

6 Envoi et réception des messages **21**

7 Implémentation **22**

8 Résultats **23**

9 Conclusion **24**

Index **25**

10 Les Exemples **28**
10.1 L'exemple 1 28
10.1.1 Principe 28
10.1.2 Sortie 29
10.2 L'exemple 2 30
10.2.1 Principe 30
10.2.2 Code 31
10.2.3 Sortie 33
10.3 L'exemple 3 34
10.3.1 Principe 34
10.3.2 Code 34
10.3.3 Sortie 35

1 Introduction

Il arrive dans le domaine de la recherche, quoique non exclusivement, qu'on bute sur les problèmes d'ordre algébrique trop complexe. Ceci est même assez fréquent dans les domaines tels que l'aérodynamique, étude de la performance des systèmes réels tels que stations service, supermarchés et chaînes de montage, ou lors des études de comportement de systèmes écologiques. Dans ce cas-là, la ***simulation*** permet d'obtenir une approximation suffisante de la réponse recherchée.

Le problème est décomposé dans un système de sous-problèmes, puis représenté par un ***modèle***. Ce modèle est ensuite traduit en algorithme informatique. Lors du calcul d'*évolution* de systèmes dynamiques complexes, en général nous distinguons l'***état*** d'un modèle à un moment précis, et son ***évolution*** dans le temps vers les états successeurs. Cette évolution peut être représenté par les équations mathématiques, ou, quand ceci n'est pas possible, par une succession d'***événements*** qui en se produisant créent d'autres événements (*simulation par événements discrets*). Dans ce dernier cas, les évènements provoquent les changements de l'état du modèle. De plus, il n'y a pas d'états intermédiaires entre deux instants discrets de simulation ($t_{i+1} = t_i + \Delta t$).

L'évolution du modèle dans le temps est représentée par une horloge. Cette horloge peut avancer de façon synchrone (Δt constant), ou de façon asynchrone (Δt variable). Lors de la simulation par événements discrets on utilisé en général une horloge asynchrone.

Si on sépare les entités du modèle, et si on les simule séparément, nous pouvons obtenir un gain de temps considérable. Dans ce cas-là, on parle de la ***simulation répartie***. Lors de la simulation repartie, les problèmes de synchronisation peuvent surgir. Du fait de la parallélisation de la simulation, chaque entité simulé possède une horloge interne propre. Si de plus, ces entités interagissent entre-eux, leur évolution dans le temps doit être synchronisé.

Deux méthodes de la synchronisation lors de la simulation répartie par événements discrets sont discutées dans ce rapport. Le modèle proposé permet de simuler de façon synchrone et asynchrone un système dynamique complexe.

2 Simulation événementielle répartie

Dans la simulation par événements discrets, les entités simulées, appelés ***Processus***, insèrent leurs ***événements*** dans un ***échéancier***. Chaque événement en s'exécutant modifie les états internes du processus simulé. Il peut aussi envoyer des messages à d'autres processus afin de simuler les interactions. Un événement possède une *estampille* qui indique sa date d'exécution. Ce mécanisme permet de gérer de façon séquentielle une file d'événements distincts.

Dans ce type de simulations, l'horloge n'est que virtuelle. Elle représente la date de dernier événement exécuté. Cette horloge n'avance pas d'elle même mais est réglé *"manuellement"* à la date du prochain événement à exécuter. Comme, par truchement de l'estampille, nous connaissons la durée d'une action, nous obtenons un modèle cohérent du temps qui s'écoule. Le temps auquel se trouve un processus s'appelle le ***temps local*** (LT: *Local Time*).

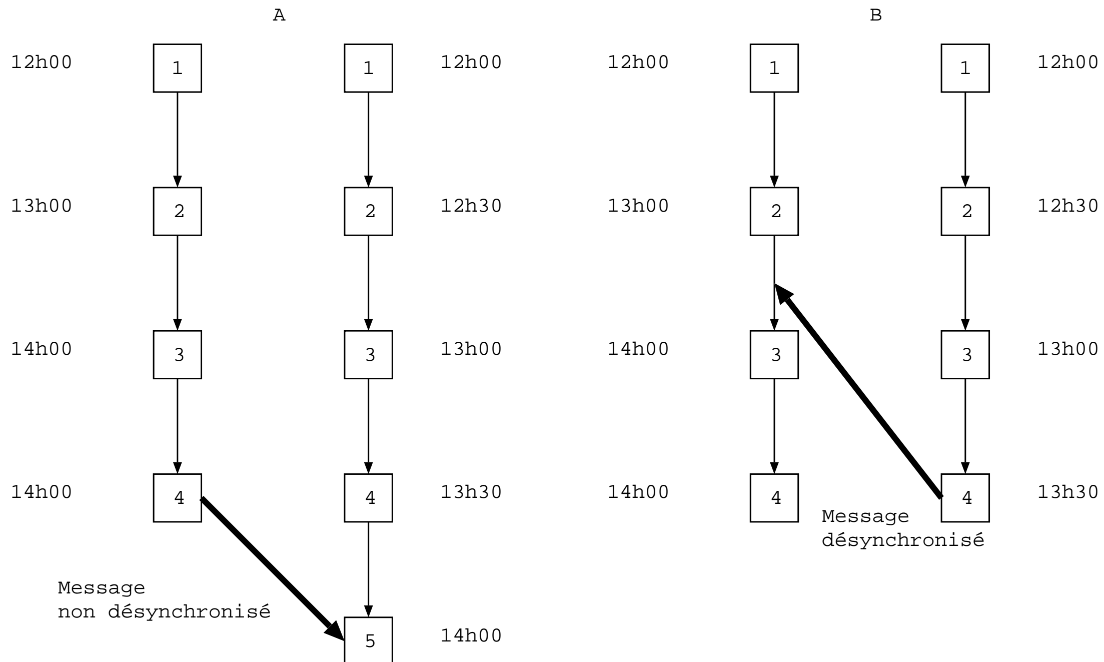
Lors de la parallélisation de la simulation événementielle dans le but de rendre son exécution plus rapide, nous devons considérer le problème suivant. Chaque processus participant à la simulation possède son *propre* temps local. Quand un processus envoie un message, le destinataire peut se trouver à une autre date. Si le destinataire est dans le passé par rapport aux expéditeur, l'événement contenu dans le message sera inscrit dans l'échéancier, puis exécuté à son tour (Fig 2.1 dessin A). Dans le cas contraire, une *incohérence temporelle* (ou désynchronisation) se produira (Fig 2.1 dessin B). Ce cas de figure est géré de deux façons différentes dans les modèles d'exécution ***pessimiste*** respectivement ***optimiste***.

2.1 Modèle d'exécution pessimiste

Dans le modèle pessimiste nous supposons que la désynchronisation temporelle arrive souvent. Pour cela, nous formons un *graphe d'exécution* à partir de notre modèle à simuler. Les noeuds de ce graphe représentent les processus, et les arcs orientés schématisent les canaux de communication unidirectionnels entre ces processus. Ainsi nous pouvons définir précisément les cheminement des messages. Nous disons que un processus P_i qui reçoit les messages d'un autre processus P_{i-1} est le processus *suivant* pour P_{i-1}. De même P_{i-1} *précède* P_i.

Un processus simulé selon le modèle pessimiste sait de qui il peut recevoir les messages (de qui il est le suivant). Pour ne pas subir de désynchronisation il attend au moins un

FIG. 2.1 –: *Exemple de désynchronisation*



message par canal de communication entrant. Ensuite il ajuste sont temps local à la date de l'événement entrant avec la date d'exécution la plus basse, puis exécute celui-ci.

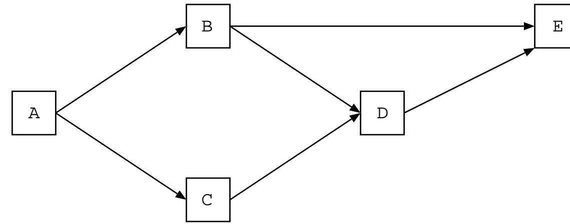
La force du modèle d'exécution pessimiste est qu'il ne peut pas être désynchronisé. Par contre, sa performance est réduite à cause de l'attente des messages entrants. De plus, si à un instant donné de la simulation un processus expéditeur n'a pas de messages à envoyer à son processus destinataire, le second attendra jusqu'au message suivant du "*processus précédent*" (précédent dans le graphe d'exécution). Ceci l'empêchera d'exécuter ses événements propres ou les évènements que les autres processus lui ont envoyé. De plus, par son inaction le processus bloquera tout le sous-graphe au delà de lui-même, car ses processus *suivants* attendront un message.

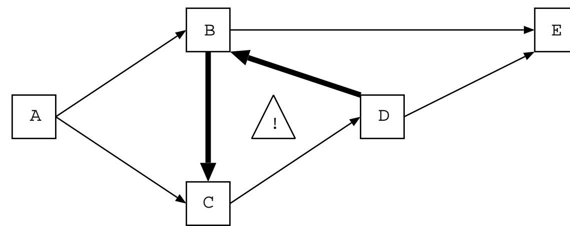
Une solution à ce problème consiste en envoi de **messages-nuls,** qui simulent l'envoi du message afin de débloquer la simulation.

2.1.1 Graphes circulaires dans le modèle d'exécution pessimiste

Le modèle d'échéancier pessimiste comporte un désavantage de taille du à sa gestion bloquante des messages. Les graphes circulaires ne sont pas gérés *de facto* par le modèle. Pour illustrer ceci consultez la figure 2.2. L'exemple comporte deux configurations de 5 processus (A, B, C, D et E). Le premier dessin illustre une simulation dans la quelle les entités échangent les messages dans un graphe orienté sans circuits(GOSC). Les processus communiqueront de gauche à droite du dessin (A,B,C,D,E ou A,C,B,D,E). Si un

FIG. 2.2 –: *Échéancier Pessimiste et graphes circulaires*





évènement doit s'exécuter en D, un message doit parvenir de B et de C. Si B ou C n'ont rien à communiquer à D, ils enverront un message-nul chacun.

Dans l'exemple du graphe orienté avec circuits, D veut envoyer un message à B. Il ne peut pas le faire tant qu'il n'a pas reçu un message de C, qui lui attend un message de A, mais de B aussi. Même si B n'a pas de message pour C, il ne peut pas envoyer de message-nul, tant que il n'a pas reçu de message de D. A peut sans façon envoyer un message-nul (Si pas d'autre message à envoyer). Le système reste bloqué.

La solution pour résoudre ce genre de problèmes consiste de définir un des échéanciers qui participent au sous-graphe circulaire en tant que échéancier asynchrone.

2.2 Modèle d'exécution optimiste

Dans le modèle optimiste, nous supposons que la désynchronisation est relativement peu fréquente. Cette vision *optimiste* des choses nous permet d'oublier la synchronisation *"dure"* du modèle pessimiste.Les événements sont exécutés sans attente. Si toutefois un message, avec l'événement dont la date d'exécution est inférieure au temps local du processus destinataire, arrive, nous devons revenir dans le temps, et ceci jusqu'à la date de l'événement reçu. Ce mécanisme est appelé le ***Rollback***. Lors du rollback tous les évènements exécutés depuis la date de désynchronisation sont annulées. Si dans ce laps de temps les messages ont été envoyés vers d'autres processus, ils sont aussi annulés,

ainsi que tous les événements que ces autres processus ont exécutés depuis la réception du messages à annuler. Une fois le rollback effectué l'exécution reprends.

Ce modèle est très efficace pour les simulations ou les processus communiquent peu. Dans le cas contraire, les rollbacks trop nombreux réduisent les performances du système. Les événements exécutés doivent être stockés dans une mémoire tampon en vue de leur annulation. Pour garder la taille de ce tampon raisonnable, il faut définir une date avant la quelle rien ne peut plus se passer. Cette date est appelle le *Temps virtuel global (GVT:Global Virtual Time)*. Le GVT est défini à tout instant de la simulation comme la date de l'événement le plus ancien, non exécute de la simulation. Cet évènement peut se trouver dans la file d'attente d'un des processus, ou encapsulé dans un message en transit. Tous les évènements de la simulation qui se trouvent dans les tampons de Rollback de leur processus respectifs en vue de leur annulation, et dont la date est ***inférieure*** à GVT, peuvent être supprimés.

Une méthode adaptative de mise à jour du GVT est discuté dans ce rapport (5.1.3).

3 But du projet

Le but de ce projet est de construire un système d' échéanciers adapté aux simulations événementielles et avec les caractéristiques suivantes:

- Simulation repartie par évènements discrets
- Modèle d'exécution synchrone et/ou asynchrone
- Modèle asynchrone optimiste et/ou pessimiste
- Système facilement distribuable sur une architecture multi-processeurs

Notre système devra gérer plusieurs échéanciers, chacun avec son temps propre.Les processus simulés devront pouvoir envoyer des messages et en recevoir. Les échéancier de différents types doivent collaborer de façon cohérente du point de vue des désynchronisations et/ou mise à jour du GVT. Les échéanciers pessimistes doivent pouvoir annuler les événements lors d'une désynchronisation, toutefois il ne généreront pas d'autres désynchronisations (ils respectent leur caractère pessimiste).

Tous les échéanciers doivent avoir la possibilité de fonctionner en mode synchrone et asynchrone. Le protocole d'envoi et de réception des messages doit pouvoir se coupler avec les différents protocoles réseau.

4 Étude préliminaire

4.1 Bases

Chaque Processus (Agent, Groupe,...) est attaché à un échéancier. Plusieurs processus peuvent être attachées au même échéancier, mais un processus appartient à un seul échéancier. Chaque échéancier possède une Boite aux lettres (BAL) dans laquelle s'inscrivent les messages qui lui sont destinés, c-à-d destinés aux processus qu'il contrôle. Chaque message encapsule un ou plusieurs événements. Ces événements sont extraits, puis insérés dans la File d'attente de l'échéancier. L'échéancier prends les événements dans la file d'attente, un par un, et les exécute sur les processus. L'échéancier peut insérer les événements dans sa propre file d'attente. Chaque événement possède une date d'exécution, ce qui permet un ordre total sur le tri de la file d'attente.

4.1.1 Définitions

Un événement est dit **sûr** s'il est impossible qu'il soit remis en cause plus tard. Un événement est dit **non sûr** s'il est possible qu'il viole la contrainte de la cohérence temporelle ou causale.

Un événement viole **la contrainte de la cohérence temporelle** ssi le message qui le déclenche porte une date d'exécution inférieure à la date courante de l'horloge de l'échéancier.

Un événement viole **la contrainte de cohérence causale** ssi il viole la contrainte de la cohérence temporelle **et** peut raisonnablement être supposé influencer les événements qui se sont produits depuis sa date d'exécution jusqu'à la date courante de l'échéancier.

On distingue le **contrôle local** et le **contrôle global**. Le contrôle local correspond aux événements qui se produisent au niveau de l'Échéancier. Le contrôle global agit au niveau de la **Simulation** (ensemble d'Échéanciers). Si la contrainte de la cohérence causale est respecté au niveau du contrôle local, elle l'est aussi au niveau du contrôle global.

4.2 Modèle d'exécution synchrone

Les événements se produisent à des intervalles de temps définis :**dt**. L'horloge avance de dt à chaque pas et dirige l'exécution des événements.

4.3 Modèle d'exécution asynchrone

Les événements s'inscrivent, avec leur date d'exécution, dans l'échéancier. L'échéancier avance son horloge jusqu'à la date du premier événement suivant. L' événement est ensuite exécuté.

4.4 Modèle asynchrone pessimiste

Le modèle asynchrone pessimiste (sûr) garantit le respect de la cohérence temporelle et de la cohérence causale et empêche toute violation de celles-ci. Pour tout message rentrant, estampillé avec l'heure locale $H_{emetteur}$, l'heure locale du processus récepteur, $H_{recepteur}$, respecte la condition: $H_{emetteur} \leq H_{recepteur}$. [1] donne les deux règles à respecter lors d'échange des messages afin de garantir la cohérence.

Règle1: Un processus attend les messages sur **tous** ses canaux d'entrée avant de sélectionner le message avec la date la plus petite.

Règle2: Un processus n'émet pas de messages estampilles avec une date inférieure à sa date locale.

Ceci implique évidemment la possibilité d'impasse. Donc un mécanisme de détection d'impasse est à mettre en oeuvre (envoi de messages vides).

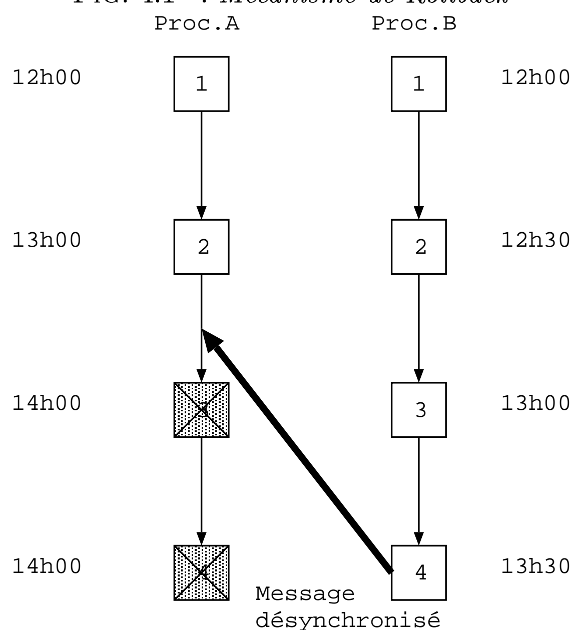
4.5 Modèle asynchrone optimiste

Le modèle asynchrone optimiste admet, dans une certaine mesure, la violation de la cohérence. Lors de la détection de la violation de la cohérence causale au niveau **local**, le retour-en-arrière (rollback) est effectué. Il est répercuté au niveau du contrôle **global** par envoi d**'anti-messages** qui annulent les messages correspondants. Il se propage vers le passé de la simulation, jusqu'à la date du message désynchronisé (violant la contrainte de cohérence). La simulation reprend à cet endroit dans le temps. Figure 4.1 montre deux processus désynchronisés et les évènement annulés lors du rollback (événements 3 et 4 du processus A).

On distingue trois types de temps. Le **temps local (LT)** indique la date à laquelle se trouve un échéancier en particulier. Le temps local peut avancer, mais aussi reculer (lors du rollback).

Le **temps virtuel local** (**LVT**, Local Virtual Time) indique la date minimum entre les événements en attente d'être exécutés et le LT. LVT est *en principe* toujours égal au LT, sauf si l'échéancier contient des messages désynchronisées dans la file d'attente. Comme les messages désynchronisés viennent avant le LT (d'où leur caractère désynchronisé), le LVT sera inférieur au LT dans ses cas là.

FIG. 4.1 –: *Mécanisme de Rollback*



Le **temps global ou GVT** (Global Virtual Time) indique la limite inférieure des dates des événements qui peuvent être remis en cause. Tout ce qui vient **avant** la date courante du GVT ne peut être défait. Le GVT et LVT sont calculés comme suit:

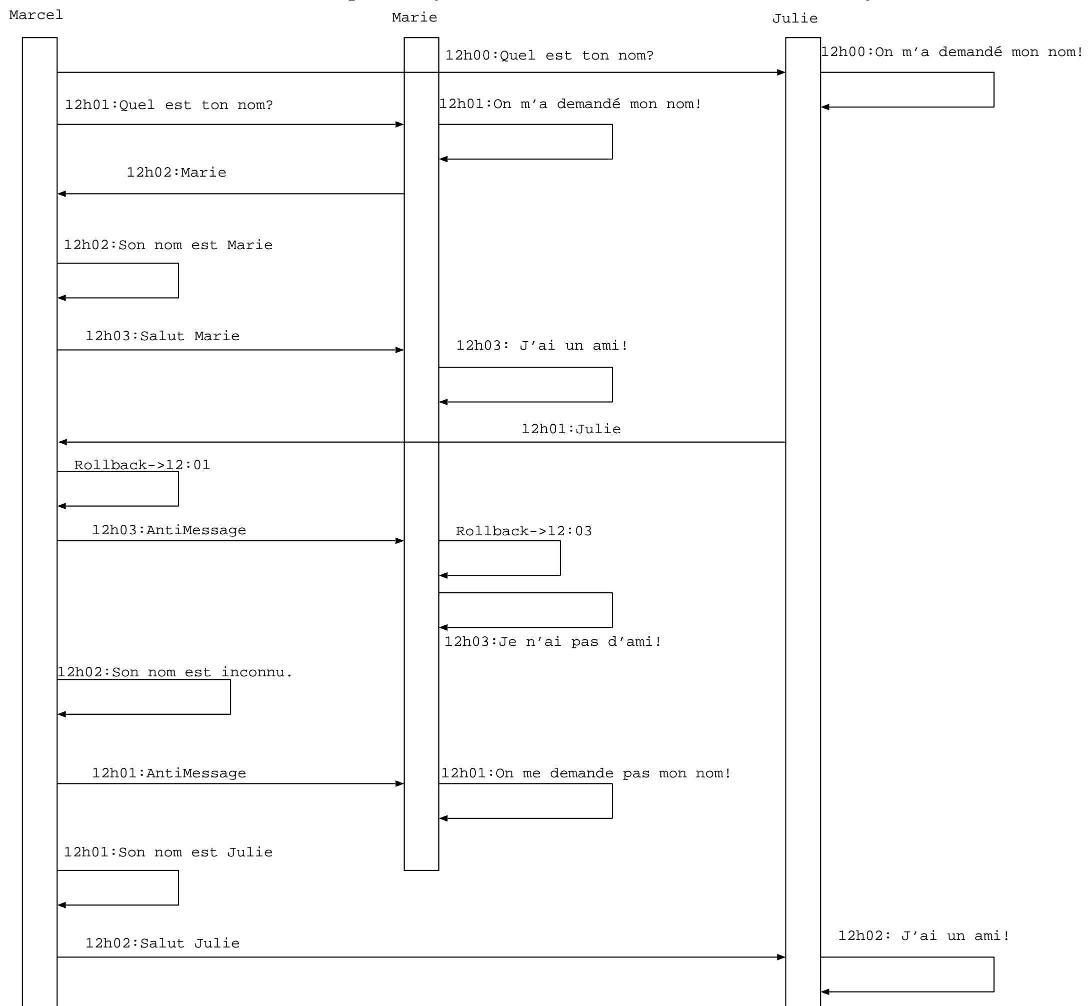
Équation 4.5:

$$GVT = \min_{\forall i} (LVT(E_i)) \text{ avec } E_i = \text{Échéancier}(i)$$

$$LVT(E_i) = min(LT, min(dates\ evts.\ en\ attente))$$

L'exemple donné par la figure 4.2 montre le fonctionnement de l'échéancier asynchrone.Trois entités y participent, Marcel, Marie et Julie. Marcel désire rencontrer une nouvelle amie. Son amie sera celle qui réponds en premier. Comme Marcel a une légère attirance purement physique envers Julie, il l'accoste en premier. Marie, ayant la détente plus rapide, réponds, sans attendre que la réponse de Julie arrive. Ayant conclu une amitié en hâte, Marcel s'aperçoit que Julie a en fait répondu la première et annule tous les liens avec Marie.

Fig. 4.2 –: *Exemple du fonctionnement de l'échéancier asynchrone*



5 Propositions de conception des échéanciers

5.1 Échéancier optimiste

5.1.1 Boucle principale

Chaque message envoyé à un échéancier, encapsule un ou plusieurs événements dans un ordre quelconque. L'échéancier asynchrone optimiste reçoit tous les messages dans sa BAL (boite au lettres)(voir fig.5.1) ou dans sa BALdeContrôle. Le ou les événements sont ensuite extraits et insérés dans la File d'attente, dans l'ordre croissant de leurs dates d'exécution pour ceux de la BAL, et en tête de file pour ceux de la BALdeContrôle. La classe Postier s'occupe d'insertion des événements dans la file d'attente. L'échéancier avance son horloge locale (LT) jusqu'à la date de l'évènement suivant, puis exécute cet événement(voir alg. 1). Ensuite, l'évènement est sauvegardé dans la liste des rollbacks ssi il peut violer la cohérence.

Si la date de l'évènement s' avère inférieure à la date de l'horloge locale, et si l'évènement en question peut violer la cohérence causale, le mécanisme de rollback est appliqué. A la fin du rollback, l'évènement resynchronisé est exécuté.

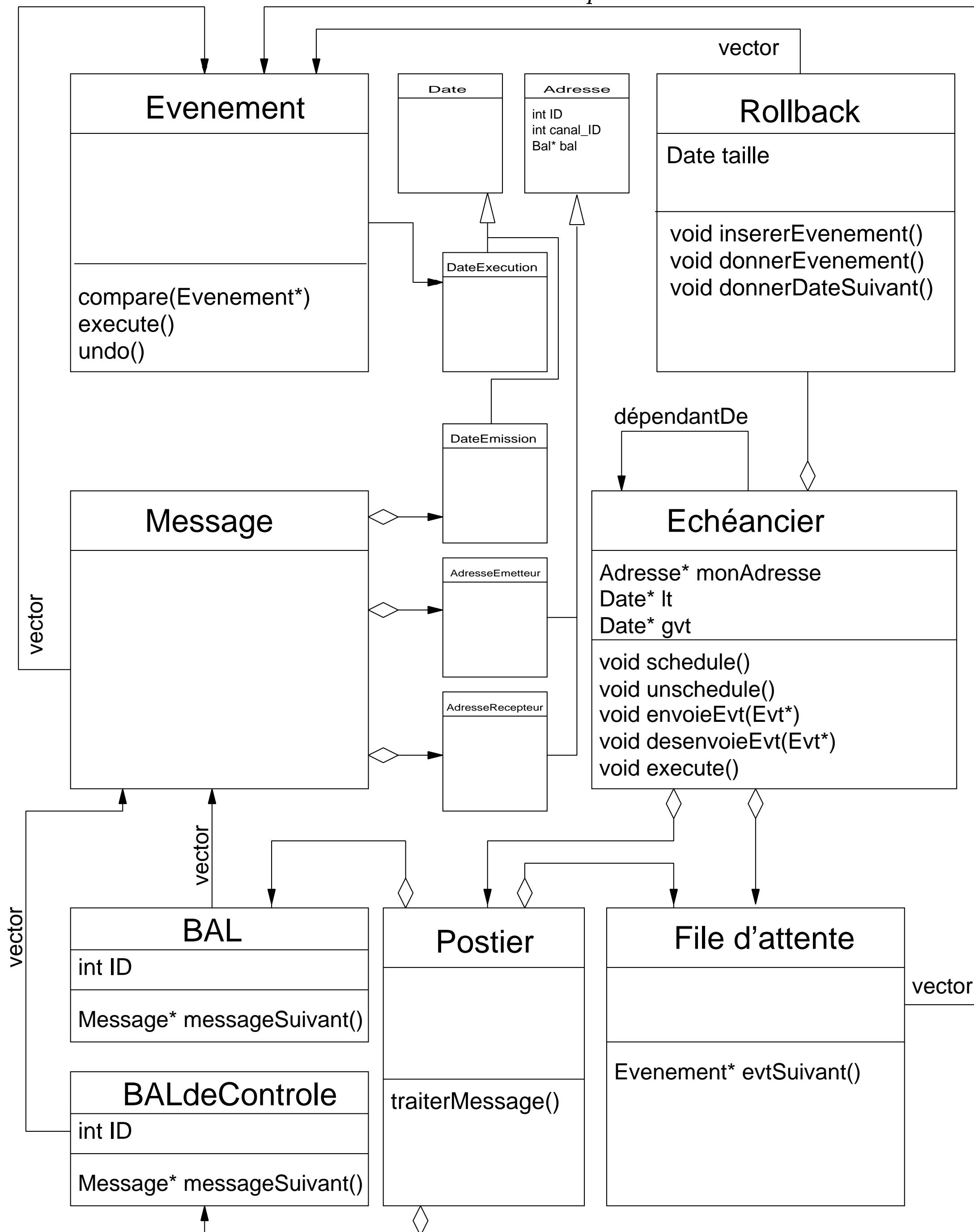
Les événements qui peuvent violer la cohérence sont ceux envoyés par un échéancier dont l'échéancier en question dépend. Si un événement ne peut pas violer la cohérence il n'est pas stocké dans le tampon des rollbacks.

5.1.2 Rollback

Lors du rollback, les événements sont pris un à un dans le tampon des rollbacks, puis annulés (voir alg.2). Le gestion du Temps virtuel global(GVT) assure qu'il ne faudra jamais annuler les événements inférieurs a GVT, ce qui borne la taille de ce tampon à LT-GVT.

L' annulation de messages envoyés à d'autres échéanciers consiste en envoi d'un anti-message dans la BAL de contrôle de l'échéancier en question. Lors de l'arrivée de l'anti-message, le postier extrait l'évènement à annuler. Si celui-ci se trouve encore dans la BAL, il est supprimé. Sinon, il est passé à l'échéancier pour l'insertion dans la file d'attente. Si l'évènement est encore dans la file d'attente de l'échéancier, il est supprimé. Sinon. l'échéancier effectue le Rollback.

FIG. 5.1 –: *Échéancier optimiste*



Algorithm 1 Algorithme général optimiste:**échéancier->lancer()**

```
pour toujours faire
    événement=fille_d'attente.prendre_événement_suivant();
    si événement->date < lt alors
        rollback();
        événement->exécuter();
    sinon
        lt.avancer_date_a(événement.date);
        événement->exécuter();
        si événement->modifie_état() alors
            tampon_rollback->sauver(événement);
        fin si
    fin si
fin pour
```

Algorithm 2 Rollback : **échéancier->rollback(date)**

```
événement=tampon_rollback->événement_suivant();
tant que événement->date_exécution()> date faire
    événement->undo();
    événement=tampon_rollback->événement_suivant();
fin tant que
tampon_rollback->insérer_événement(événement);
```

5.1.3 Gestion du Temps virtuel global (GVT)

La gestion du GVT doit assurer que l'intervalle LT-GVT ne dépasse jamais une certaine valeur, qui correspond à la limite de la taille du Tampon des Rollbacks.

Au départ le tampon de rollback est initialisé à une valeur choisie avec attention. Les événements exécutés sont insérés dans le tampon. Si pour un échéancier la limite de stockage est atteinte, il effectue la demande de mise-à-jour du GVT. Cette demande consiste à envoyer à tous les échéanciers un message demandant le LVT. Les échéanciers répondent avec leur LVT. L'échéancier qui a effectué la demande calcule le minimum et le renvoie à tous les autre échéanciers (3*N messages). Ensuite il positionne son GVT à cette date minimum, il efface tous les événements antérieurs qui se trouvent dans le tampon des rollbacks, et il augmente la taille du tampon (voir alg.3).

Algorithm 3 Stockage des événements pour le rollback et mise-à-jour du GVT:**tampon_rollback->sauver(événement)**

```
si disponible_place() alors
   sauver(événement);
sinon
   diffuser(message_contrôle.demande_LVT);
   attendre_et_récupérer(LVT(i));
   GVT=calculer_minimum(LVT(i));
   diffuser(message_contrôle.mise_a_jour_GVT(GVT));
   supprimer_antérieur_a(GVT);
   augmenter_taille();
   sauver(événement);
fin si sinon
```

Quand un échéancier reçoit la demande du LVT, puis le GVT en réponse, il met ce dernier comme le nouveau GVT. Il enlève tous les événements antérieurs à cette date et diminue la taille du tampon des rollbacks.

Le problème qui se pose est celui d'un échéancier qui est dans le futur lointain par rapport aux autres et qui ne reçoit pas de messages (ces messages seraient désynchronisées, et le ramènerait dans le passé (rollback)). Sa file d'attente risque de grandir à l'infini vu que toutes les mises à jour du GVT sont dans le passé. Il existent 3 solutions:

1. Rendre pessimiste cet échéancier dès le départ.
2. Trouver un algorithme qui le passerait en pessimiste en moment voulu (puis de nouveau en optimiste, si nécessaire).
3. Stopper l'exécution (unschedule) de l'échéancier tant que la taille du tampon des rollbacks ne descend pas. (tant que les autres ne le "rattrapent" pas).
4. Effectuer la mise à jour du GVT de façon synchrone, à des intervalles prédéfinis.

5.1.4 Gestion de la taille du tampon des rollbacks

Il s'agit de borner la taille du tampon des rollbacks (TdR) afin de minimiser l'utilisation de la mémoire. Dans notre modèle, ceci est fait d'une manière dynamique (voir sec.5.1.3).

Exemple:

Lors de l'augmentation de la taille du tampon, on augmente celui-ci de moitié de sa taille précédente.

Lors de la réduction de celui-ci, on réduit la taille de la moitié de l'espace inutilisé.

5.2 Échéancier pessimiste

L'échéancier pessimiste est dans ce modèle un cas particulier de l'échéancier optimiste. Comme il communique avec les échéanciers optimistes, il doit pouvoir effectuer les rollbacks. Toutefois il ne peut pas provoquer de rollbacks grâce à sa gestion , différente, des événements à exécuter(voir sec.4.4). Il dépend, par contre, des autres échéanciers. L'utilisateur doit définir sur quels autres échéanciers l'échéancier doit attendre pour pouvoir avancer. Les échéanciers dont un autre échéancier dépend, ne doivent **jamais** envoyer les messages avec la dates d'exécution **supérieure** à leur LT (voir sec.4.4). Algorithme 4 donne la procédure principale.

Algorithm 4 Algorithme général pessimiste**:échéancier->lancer()**

```
pour toujours faire
    tant que (!tout_message_reçu()) attendre;
    événement=file_d'attente.événement_suivant();
    lt.avancer_date_a(événement.date);
    événement->exécuter();
    si événement->dépendant_causallement alors
        tampon_rollback->sauver(événement);
    fin si
fin pour
```

Pour pouvoir exécuter un événement sur un échéancier pessimiste, nous devons d'abord nous assurer que les messages des échéanciers dont l'échéancier traité dépend sont tous arrivées.

5.3 Échéancier synchrone

Dans ce modèle l'échéancier synchrone est un cas particulier de l'échéancier asynchrone. Il doit pouvoir effectuer les rollbacks et les provoquer. En plus il a une procédure qui

doit être exécuté à des intervalles de temps constants, **DT**. L'algorithme exécute les événements asynchrones selon le modèle de l'échéancier optimiste jusqu'à la date du prochain événement synchrone. Il exécute celui-ci, puis continue (voir alg.5).

Algorithm 5 Algorithme général synchrone: **échéancier->lancer(DT)**

```
pour toujours faire
    si date_asynchrone_suivant < date_synchrone_suivant alors
       événement=fille_d'attente.prendre_événement_suivant();
       si événement->date < lt alors
             rollback();
             événement->exécuter();
       sinon
          lt.avancer_date_a(événement.date);
          événement->exécuter();
          si événement->modifie_état() alors
             tampon_rollback->sauver(événement);
          fin si
      fin si sinon
    sinon
      lt.avancer_date_a(événement_synchrone->date)
      événement_synchrone->exécuter(date_synchrone_suivant);
      date_synchrone_suivant=date_synchrone_suivant+DT
      si événement_synchrone->modifie_état alors
             tampon_rollback->sauver(événement_synchrone);
      fin si
    fin si sinon
fin pour
```

Une autre possibilité pour adresser ce problème existe. On peut rendre l'évènement lui même synchrone. Dans ce cas, l'échéancier asynchrone doit être légèrement modifié afin qu'il puisse accepter les évènements synchrones. S'il extrait un évènement synchrone de sa file d'attente, il doit, après l'avoir exécuté, réinsérer une copie de cet évènement dans la file à la date indique par la date courante plus la valeur de la période de l'évènement synchrone exécuté. Cette approche supprime la nécessité d'avoir un échéancier synchrone et nous l'avons utilisé dans l'implémentation du prototype.

6 Envoi et réception des messages

La classe Postier s'occupe d'envoyer et recevoir les messages. Lors de la procédure d'envoi de message, l'échéancier envoie son message à la BAL de l'adresse. Comme chaque adresse contient une référence sur la BAL, ceci est fait très simplement.

7 Implémentation

cf. documentation au format html et l'Annexe.

8 Résultats

Dans les petits exemples qui ont été utilisés, le prototype semble bien se comporter. La collaboration entre les échéanciers pessimistes et optimistes n'induit aucune incohérence constatable avec le jeux de tests utilisé. Toutefois, les tests "grandeur nature" devraient être effectués. Les tests sont décrits dans les exemples de l'annexe.

La gestion du tampon de rollback est assez efficace. Après 1000 tours de l'exemple 3 la taille de ce dernier stagne vers 35, ce qui est acceptable si on considère que plus que six mille événements ont été traités et le même nombre de messages envoyés. L'occupation en mémoire de la simulation reste donc stable.

Le prototype est toutefois très lent. Un ordinateur équivalent à un Intel Pentium IV 2 Ghz (platforme de test), n'est pas un luxe. La création dynamique de beaucoup d'objets (évènements, messages) fait saturer la machine virtuelle java qui elle, prends toutes les ressources processeur. Il est toutefois possible d'exécuter une large simulation jusqu'au bout.

Les tests on été faits sur une machine Linux. Dans notre exemple 16 threads (natifs) ont été crées durant l'exécution de la simulation. La platforme de test à tenu la route et restait disponible pour un autre travail avec la simulation en arrière plan. Aucun test n'a été effectué sur les platformes MSWindows ou MacOS?. Nous avons été capables de lancer simultanément jusqu'au 10 instances du code de l'exemple 3 (160 threads, 60'000 messages) et constater que en 10 minutes on arrivé au tour 46-48 de la simulation et ceci pour toutes les instances du code. Avec une seule instance du code exécuté durant la même période de temps nous arrivons jusqu'au tour 295 environ.

Le prototype manque cruellement d'une tactique efficace de la gestion des sorties sur l'affichage. Actuellement le tout est affiché, y compris les évènements qui seront annulés par la suite. Il faudrait, au contraire n'afficher que les sorties des processus simulées qui se sont produites jusqu'au GVT. Nous obtenons ainsi un affichage qui suit le déroulement *effectif* de la simulation.

9 Conclusion

Notre prototype a montré que la collaboration des échéanciers de différents types pouvait être géré par l'application de règles simples. Il est très aisé d'imaginer un échéancier asynchrone pessimiste comme un échéancier optimiste avec un clause sur l'exécution de l'événement suivant. Cette clause n'est autre que l'état des canaux d'entrée.

Le cas de l'échéancier synchrone est encore plus simple. C'est l'événement lui même qui est rendu synchrone. Un événement synchrone n'est rien d'autre qu'un événement générique qui en plus, une fois son code exécuté, inscrit une copie de lui-même à la date donné par l'addition de la date courante avec la valeur de la période.

Le fait de ramener tous les échéanciers au leur dénominateur commun, ce qui est indiqué à cause du besoin de tampon de Rollback pour tous, supprime une bonne partie des problèmes.

La stabilité au point de vue occupation en mémoire est un autre point important. Une fois les problèmes de création d'objets java résolus, le système pourrait être applique à des simulations portant sur un grand nombre d'entités.

Index

Échéancier, 19
échéancier, 6
écheancier pessimiste, 19
événement, 5, 6

anti-message, 12
asynchrone, 12

cohérence causale, 11
cohérence temporelle, 11
contrôle global, 11
contrôle local, 11

graphe d'exécution, 6
GVT, 9, 13, 18

LT, 6, 12
LVT, 12

message, 21
modèle d'exécution optimiste, 6, 8
modèle d'exécution pessimiste, 6

Processus, 6, 11

rollback, 8, 19
rRollback, 15

simulation, 5, 11
simulation répartie, 5
synchrone, 19

temps local, 6, 12
temps virtuel global, 9, 18
temps virtuel local, 12

Bibliographie

[1] Pontien Déguénon. *Modèle d'agent pour la simulation répartie par événements discrets*. PhD thesis, Université de Neuchâtel, 1996.

Annexe

10 Les Exemples

10.1 L'exemple 1

10.1.1 Principe

Dans cet exemple nous avons 6 échéanciers A,B,C,D,E et F, qui contrôlent deux équipes de 3 coureurs.

La première: Alice, Bernard et Claude. (Échéanciers A, B respectivement C)

La deuxième: Daniel, Edgar et Fabian. (Échéanciers D, E respectivement F)

A et D commencent a courir et à la fin de leur course passent l'estafette aux leurs successeurs B respectivement E, et ainsi de suite. Les échéanciers sont pessimistes, donc il n'y aura pas de désynchronisation. Observez les dépendances entre les coureurs et celles entre les échéanciers (cf code source plus bas).

```
public class Estaffette {
 public static void main(String [] args){
 //Equipe 1
 EcheancierPessimiste echA=new EcheancierPessimiste("A");
 EcheancierPessimiste echB=new EcheancierPessimiste("B");
 EcheancierPessimiste echC=new EcheancierPessimiste("C");
 Courreur alice=new Courreur("Alice");
  alice.vitesse=10;
  alice.distance=50;
  alice.setEcheancier(echA);
Courreur bernard=new Courreur("Bernard");
  bernard.vitesse=10;
  bernard.distance=50;
  bernard.setEcheancier(echB);
 Courreur claude=new Courreur("Claude");
  claude.vitesse=10;
  claude.distance=50;
  claude.setEcheancier(echC);
alice.suivant=bernard;
 bernard.suivant=claude;
claude.suivant=null;
 echC.dependsDe(echB);
 echB.dependsDe(echA);
 echA.dontDepends(echB);
 echB.dontDepends(echC);
//Fin Equipe 1
```

```
 //Equipe 2
 EcheancierPessimiste echD=new EcheancierPessimiste("D");
 EcheancierPessimiste echE=new EcheancierPessimiste("E");
 EcheancierPessimiste echF=new EcheancierPessimiste("F");
 Courreur daniel=new Courreur("Daniel");
  daniel.vitesse=10;
  daniel.distance=50;
  daniel.setEcheancier(echD);
Courreur edgar=new Courreur("Edgar");
  edgar.vitesse=10;
  edgar.distance=50;
  edgar.setEcheancier(echE);
 Courreur fabian=new Courreur("Fabian");
  fabian.vitesse=10;
  fabian.distance=50;
  fabian.setEcheancier(echF);
daniel.suivant=edgar;
 edgar.suivant=fabian;
fabian.suivant=null;
 echF.dependsDe(echE);
 echE.dependsDe(echD);
 echD.dontDepends(echE);
 echE.dontDepends(echF);
//Fin Equipe 2
//ici on fait courrir Alice et Daniel
 EV_Cours ev1=new EV_Cours(new java.util.Date());
 ev1.setProcessus(alice);
 echA.insereEvt(ev1);
 EV_Cours ev2=new EV_Cours(new java.util.Date());
 ev2.setProcessus(daniel);
 echD.insereEvt(ev2);
 /*
 Insertion des echeanciers dans la simulation.
 Attention, les echeanciers seront lancés dans l'ordre dans lequel
 ils sont inséres.
*/
 Simulation.add(echA);
 Simulation.add(echB);
 Simulation.add(echC);
 Simulation.add(echD);
 Simulation.add(echE);
 Simulation.add(echF);
 //On lance la Simulation
 Simulation.start();
 }
}
```

10.1.2 Sortie

Les informations à droite sont inscrites par les échéanciers et celles à gauche par les entités simulés.

java demos/estafette/Estaffette

(A): Attends les dependances...

(A): ...Ok
(B): Attends les dependances...
(C): Attends les dependances...
(D): Attends les dependances...
(D): ...Ok
(E): Attends les dependances...
(F): Attends les dependances...
A: Mon Jun 10 18:18:58 CEST 2002

Alice: Je cours.
Alice:heure= Mon Jun 10 23:18:58 CEST 2002

D: Mon Jun 10 18:18:58 CEST 2002

Daniel: Je cours.
Daniel:heure= Mon Jun 10 23:18:58 CEST 2002

(D): Attends les dependances...
(D): ...Ok
(D): unscheduled
(A): Attends les dependances...
(A): ...Ok
(A): unscheduled
(B): ...Ok
B: Mon Jun 10 23:18:58 CEST 2002

Bernard: Je cours.
Bernard:heure= Tue Jun 11 04:18:58 CEST 2002

(B): Attends les dependances...
(E): ...Ok
E: Mon Jun 10 23:18:58 CEST 2002

Edgar: Je cours.
Edgar:heure= Tue Jun 11 04:18:58 CEST 2002

(E): Attends les dependances...
(F): ...Ok
F: Tue Jun 11 04:18:58 CEST 2002

Fabian: Je cours.
Fabian:heure= Tue Jun 11 09:18:58 CEST 2002

(F): Attends les dependances...
(C): ...Ok
C: Tue Jun 11 04:18:58 CEST 2002

Claude: Je cours.
Claude:heure= Tue Jun 11 09:18:58 CEST 2002

(C): Attends les dependances...

10.2 L'exemple 2

10.2.1 Principe

Dans cet Exemple l'équipe d'Alice est optimiste, et celle de Daniel pessimiste. De plus Alice fait la maligne et prévoie de courir avant le coup du Pistolet. Celui-ci, une fois déclenché, désynchronise Alice.

Veuillez vous référer au code source pour les indications. Quelques remarques importantes sont formulées dans le code source, celui-ci étant assez lisible pour que on s'y réfère.

10.2.2 Code

```
 public static void main(String [] args){
 //Equipe 1
 EcheancierOptimiste echA=new EcheancierOptimiste("A");
 EcheancierOptimiste echB=new EcheancierOptimiste("B");
 EcheancierOptimiste echC=new EcheancierOptimiste("C");
 Courreur alice=new Courreur("Alice");
  alice.vitesse=10;
  alice.distance=50;
  alice.setEcheancier(echA);
Courreur bernard=new Courreur("Bernard");
  bernard.vitesse=10;
  bernard.distance=50;
  bernard.setEcheancier(echB);
 Courreur claude=new Courreur("Claude");
  claude.vitesse=10;
  claude.distance=50;
  claude.setEcheancier(echC);
alice.suivant=bernard;
 bernard.suivant=claude;
claude.suivant=null;
//Fin Equipe 1
 //Equipe 2
 EcheancierPessimiste echD=new EcheancierPessimiste("D");
 EcheancierPessimiste echE=new EcheancierPessimiste("E");
 EcheancierPessimiste echF=new EcheancierPessimiste("F");
 Courreur daniel=new Courreur("Daniel");
  daniel.vitesse=10;
  daniel.distance=50;
  daniel.setEcheancier(echD);
Courreur edgar=new Courreur("Edgar");
  edgar.vitesse=10;
  edgar.distance=50;
  edgar.setEcheancier(echE);
 Courreur fabian=new Courreur("Fabian");
  fabian.vitesse=10;
  fabian.distance=50;
  fabian.setEcheancier(echF);
daniel.suivant=edgar;
 edgar.suivant=fabian;
fabian.suivant=null;
 echF.dependsDe(echE);
 echE.dependsDe(echD);
 echD.dontDepends(echE);
 echE.dontDepends(echF);
//Fin Equipe 2
 Pistolet p1=new Pistolet();
 EV_Depart evD=new EV_Depart(new java.util.Date());
 evD.setProcessus(p1);
 //On peut ajouter les processus dependants de l'evenement
 //directement dans l'evenement. Pourquoi s'en priver?
 //Celà-dit, je n'affirme en aucune manière que c'est propre..
 evD.addCourreur(alice);
 evD.addCourreur(daniel);
 EcheancierOptimiste echPistolet=new EcheancierOptimiste("P:");
 p1.setEcheancier(echPistolet);
```

```
echPistolet.insereEvt(evD);
 //Comme Daniel est pessimiste, il doit dependre du pistolet
 echD.dependsDe(echPistolet);
 echPistolet.dontDepends(echD);
//l'equipe d'alice decide de prendre l'initiative ... ils courrent deja, les goujats.
 EV_Cours ev1=new EV_Cours(new java.util.Date());
 ev1.setProcessus(alice);
 echA.insereEvt(ev1);
/*
 Insertion des echeanciers dans la simulation.
 Attention, les echeanciers seront lancés dans l'ordre dans lequel
 ils sont inséres.
*/
 //Simulation.add(echPistolet); // si on met le depart de la course ici,
 // A B et C courrent 2 fois.
 Simulation.add(echA);
 Simulation.add(echPistolet); // si on met le depart de la course ici,
 // A B et C courrent 2 fois, mais ils seront
 // desynchronisés.(défois C ne courre meme pas)
 Simulation.add(echB);
 Simulation.add(echC);
 Simulation.add(echD);
 Simulation.add(echE);
 Simulation.add(echF);
 //Lancement de la simulation
 Simulation.start();

 /** Petit amusement:
 Faites courrir Alice avec une vitesse negative. Elle courera vers le
 passé, donc son equipe arrivera avant celle de Daniel.
 Explication de l'effet de bord:
 Si on courre 50km dans le sens positif et avec la vitesse negative,
 on se deplace dans le passé! C'est fou la simulation...
 Il n'y aura pas de désynchronisation supplementaire à cause de cela, car
 avant l'execution du premier événement de l'écheancier, le temps
 n'existe pas. (en fait il est a 0)
 Si Bernard avait deja fait qqchose avant de recevoir le message du
 passé d'alice, il se serait desynchronisé.
 Amusemment suite II :
 --------------------------
 Essayez de mettre Bernard en Pessimiste.
 Indication:
 n'oubliez pas les dependances!
 echA.dontDepends(echB);
 echB.dependsDe(echA);
 Resultat, Bernard attends Alice, mais doit faire le rollback à cause
 de l'erreur de Alice
 Amusemment suite II :
 --------------------------
 Faites Alice Pessimiste. Noubliez pas:
  echPistolet.dontDepends(echA);
  echA.dependsDe(echPistolet);
 Alice DECIDE de courrir avant, mais attends le coup du pistolet.
 Elle part quand même en avance et son équipe arrive la premiere,
 (de quelques nanosecondes, certes..) et ceci parce que elle n'est
 pas partie au temps doné par le pistolet (Attends les dependances..),
 mais au temps de son choix.
```

```
 Comme quoi, l'anticipation est la mère d'une bonne course..
**/

 }
}
```

10.2.3 Sortie

Observez la suppression de l'ordre de courir de la liste d'attente de l'échéancier B (Bernard), due au rollback. En effet, Alice se précipite, fait un tour et passe l'estafette à Bernard. Celui-ci n'a pas le temps de réagir et le *faux* message est supprimé par l'anti-message de Alice. Même si sur l'affichage Alice semble commencer la course après le coup du pistolet, il faut comprendre qu'elle cours avant d'avoir reçu le message de courir. Les dates correspondent car et le pistolet et Alice démarrent la simulation au même moment.

```
java demos/estafette2/Estaffette
                                                  (E): Attends les dependances...
                                                  (D): Attends les dependances...
                                                  (C): unscheduled
                                                  (B): unscheduled
                                                  (F): Attends les dependances...
                                                  P:: Mon Jun 10 18:22:19 CEST 2002
                                                  A: Mon Jun 10 18:22:19 CEST 2002
Pistolet: DEPART.
                                                  (P:): unscheduled
Alice: Je cours.
Alice:heure= Mon Jun 10 23:22:19 CEST 2002
                                                  (B): scheduled
                                                  (A): debut Rollback
Alice: Oups, je n'ai jamais couru...
                                                  (B): Message suprime de la file
                                                  (A): fin Rollback
                                                  A: Mon Jun 10 18:22:19 CEST 2002
Alice: Je cours.
Alice:heure= Mon Jun 10 23:22:19 CEST 2002
                                                  (A): unscheduled
                                                  (D): ...Ok
                                                  B: Mon Jun 10 23:22:19 CEST 2002
Bernard: Je cours.
Bernard:heure= Tue Jun 11 04:22:19 CEST 2002
                                                  (C): scheduled
                                                  (B): unscheduled
                                                  D: Mon Jun 10 18:22:19 CEST 2002
Daniel: Je cours.
Daniel:heure= Mon Jun 10 23:22:19 CEST 2002
                                                  (D): Attends les dependances...
                                                  (E): ...Ok
                                                  C: Tue Jun 11 04:22:19 CEST 2002
Claude: Je cours.
Claude:heure= Tue Jun 11 09:22:19 CEST 2002
                                                  (C): unscheduled
                                                  E: Mon Jun 10 23:22:19 CEST 2002
```

Edgar: Je cours.
Edgar:heure= Tue Jun 11 04:22:19 CEST 2002

(E): Attends les dependances...
(F): ...Ok
F: Tue Jun 11 04:22:19 CEST 2002

Fabian: Je cours.
Fabian:heure= Tue Jun 11 09:22:19 CEST 2002

(F): Attends les dependances...

10.3 L'exemple 3

10.3.1 Principe

Cet exemple est identique au précèdent, à la seule différence qu'on cours plusieurs tours. Ceci permet de rendre compte de la gestion du tampon des rollbacks. Observez le code source pour les indications.

10.3.2 Code

```
public class Estaffette {
 public static void main(String [] args){
 //Équipe 1
 EcheancierOptimiste echA=new EcheancierOptimiste("A");
 //pour voir la gestion de l'espace du tampon du rollback et avance du GVT
 echA.setTailleRollback(3); //Par defaut à 10
 EcheancierOptimiste echB=new EcheancierOptimiste("B");
 EcheancierOptimiste echC=new EcheancierOptimiste("C");
 Courreur alice=new Courreur("Alice");
  alice.vitesse=10;
  alice.distance=50;
  alice.setEcheancier(echA);
Courreur bernard=new Courreur("Bernard");
  bernard.vitesse=10;
  bernard.distance=50;
  bernard.setEcheancier(echB);
 Courreur claude=new Courreur("Claude");
  claude.vitesse=10;
  claude.distance=50;
  claude.setEcheancier(echC);
alice.suivant=bernard;
 bernard.suivant=claude;
claude.suivant=alice;
alice.nbTours=10;
bernard.nbTours=10;
claude.nbTours=10;
//Fin Equipe 1
(... idem que l'exemple 2...)
 //Lancement de la simulation
 Simulation.start();
 }
}
```

10.3.3 Sortie

Ce qui est intéressant dans cet exemple est la modification de la taille du tampon de rollback. Dans l'implémentation actuelle de l'algorithme, on augmente la taille du tampon de un et on descends de deux (cf. Section 5.1.3 p.18). Alice commence avec un tampon de 3 et les autres de 10. Une fois la limite du tampon atteinte, il augmente à 4. Alice récupère les LVTs, calcule le GVT et réduit son tampon à 2 (4 moins 2). Elle émet le nouveau GVT, ce qui provoque la diminution de tampon à 8 chez les autres coureurs.

```
                                                  (B): unscheduled
                                       (E): Attends les dependances...
                                       (D): Attends les dependances...
                                                          (D): ...Ok
                                                  (D): unscheduled
                                                  (C): unscheduled
                                       (F): Attends les dependances...
                                   P:: Mon Jun 10 18:32:12 CEST 2002
Pistolet: DEPART.
                                                    (D): scheduled
                                                 (P:): unscheduled
                                    A: Mon Jun 10 18:32:12 CEST 2002
Alice: Je cours le tour no: 0
Alice:heure= Mon Jun 10 23:32:12 CEST 2002
                                                    (B): scheduled
                                                (A): debut Rollback
Alice: Oups, je n'ai jamais couru...
                                      (B): Message suprime de la file
                                                  (A): fin Rollback
                                    A: Mon Jun 10 18:32:12 CEST 2002
Alice: Je cours le tour no: 0
Alice:heure= Mon Jun 10 23:32:12 CEST 2002
                                                  (A): unscheduled
                                       (D): Attends les dependances...
                                    B: Mon Jun 10 23:32:12 CEST 2002
Bernard: Je cours le tour no: 0
Bernard:heure= Tue Jun 11 04:32:12 CEST 2002
                                                    (C): scheduled
                                                  (B): unscheduled
                                    C: Tue Jun 11 04:32:12 CEST 2002
Claude: Je cours le tour no: 0
Claude:heure= Tue Jun 11 09:32:12 CEST 2002
                                                    (A): scheduled
                                                  (C): unscheduled
                                    A: Tue Jun 11 09:32:12 CEST 2002
Alice: Je cours le tour no: 1
Alice:heure= Tue Jun 11 14:32:12 CEST 2002
                                                    (B): scheduled
                                          A->RBack: Taille:4(monte)
                                       A->RBack: Taille:2(descends)
                              (A):GVT: Mon Jun 10 18:32:12 CEST 2002
     (A): Mise-a-jour du GVT des autres: Mon Jun 10 18:32:12 CEST 2002
                                      P:->RBack: Taille:8(descends)
                             (P:):GVT: Mon Jun 10 18:32:12 CEST 2002
                                       B->RBack: Taille:8(descends)
                              (B):GVT: Mon Jun 10 18:32:12 CEST 2002
```

```
                                   C->RBack: Taille:8(descends)
                     (C):GVT: Mon Jun 10 18:32:12 CEST 2002
                                   D->RBack: Taille:8(descends)
                     (D):GVT: Mon Jun 10 18:32:12 CEST 2002
                                   E->RBack: Taille:8(descends)
                     (E):GVT: Mon Jun 10 18:32:12 CEST 2002
                                   F->RBack: Taille:8(descends)
                     (F):GVT: Mon Jun 10 18:32:12 CEST 2002
                                               (A): unscheduled
                                               (C): unscheduled
                          B: Tue Jun 11 14:32:12 CEST 2002
```

Bernard: Je cours le tour no: 1
Bernard:heure= Tue Jun 11 19:32:12 CEST 2002

(C): scheduled
(B): unscheduled
(P:): unscheduled
C: Tue Jun 11 19:32:12 CEST 2002

Claude: Je cours le tour no: 1
Claude:heure= Wed Jun 12 00:32:12 CEST 2002

(A): scheduled
(C): unscheduled
A: Wed Jun 12 00:32:12 CEST 2002

```
(...)
```

Claude: Je cours le tour no: 3
Claude:heure= Thu Jun 13 06:32:12 CEST 2002

(A): scheduled
(C): unscheduled
A: Thu Jun 13 06:32:12 CEST 2002

Alice: Je cours le tour no: 4
Alice:heure= Thu Jun 13 11:32:12 CEST 2002

(B): scheduled
(A): unscheduled
B: Thu Jun 13 11:32:12 CEST 2002

Bernard: Je cours le tour no: 4
Bernard:heure= Thu Jun 13 16:32:12 CEST 2002

(C): scheduled
(B): unscheduled
C: Thu Jun 13 16:32:12 CEST 2002

```
(...)
```

Alice: Je cours le tour no: 7
Alice:heure= Sat Jun 15 08:32:12 CEST 2002

(B): scheduled
(A): unscheduled
B: Sat Jun 15 08:32:12 CEST 2002

Bernard: Je cours le tour no: 7
Bernard:heure= Sat Jun 15 13:32:12 CEST 2002

(C): scheduled

```
                                     B->RBack: Taille:9(monte)
                                   B->RBack: Taille:7(descends)
                     (B):GVT: Mon Jun 10 18:32:12 CEST 2002
(B): Mise-a-jour du GVT des autres: Mon Jun 10 18:32:12 CEST 2002
                                   A->RBack: Taille:0(descends)
                     (A):GVT: Mon Jun 10 18:32:12 CEST 2002
                                  P:->RBack: Taille:6(descends)
                    (P:):GVT: Mon Jun 10 18:32:12 CEST 2002
                                   C->RBack: Taille:6(descends)
```

```
(C):GVT: Mon Jun 10 18:32:12 CEST 2002
        D->RBack: Taille:6(descends)
(D):GVT: Mon Jun 10 18:32:12 CEST 2002
        E->RBack: Taille:6(descends)
(E):GVT: Mon Jun 10 18:32:12 CEST 2002
        F->RBack: Taille:6(descends)
(F):GVT: Mon Jun 10 18:32:12 CEST 2002
                       (B): unscheduled
                      (P:): unscheduled
                       (A): unscheduled
```

C: Sat Jun 15 13:32:12 CEST 2002

Claude: Je cours le tour no: 7
Claude:heure= Sat Jun 15 18:32:12 CEST 2002

(A): scheduled
(C): unscheduled
A: Sat Jun 15 18:32:12 CEST 2002

```
(...)
```