Projet CSC4102: Simulation de programmes pour la détection d'interblocage

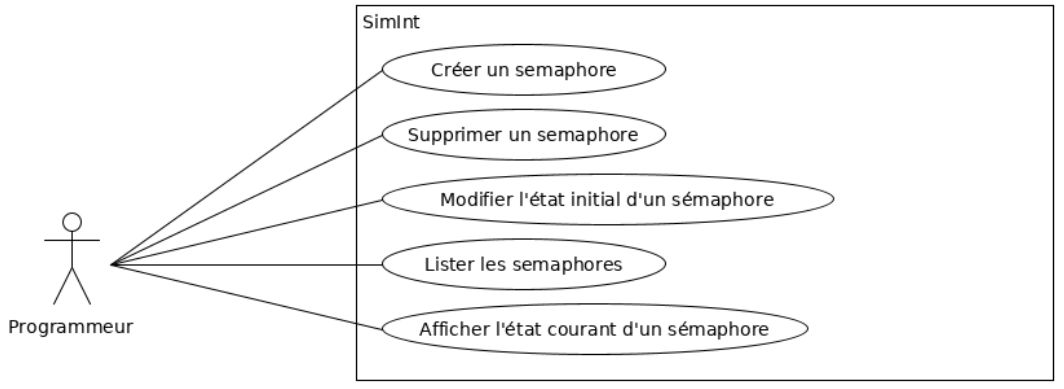
CHAFFARDON Pierre et DENIZE Julien

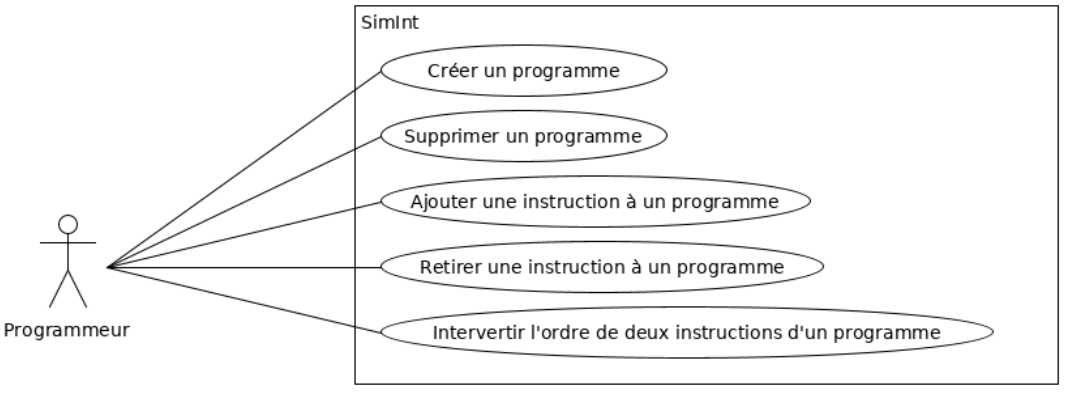
Année 2018–2019—2 Avril 2019

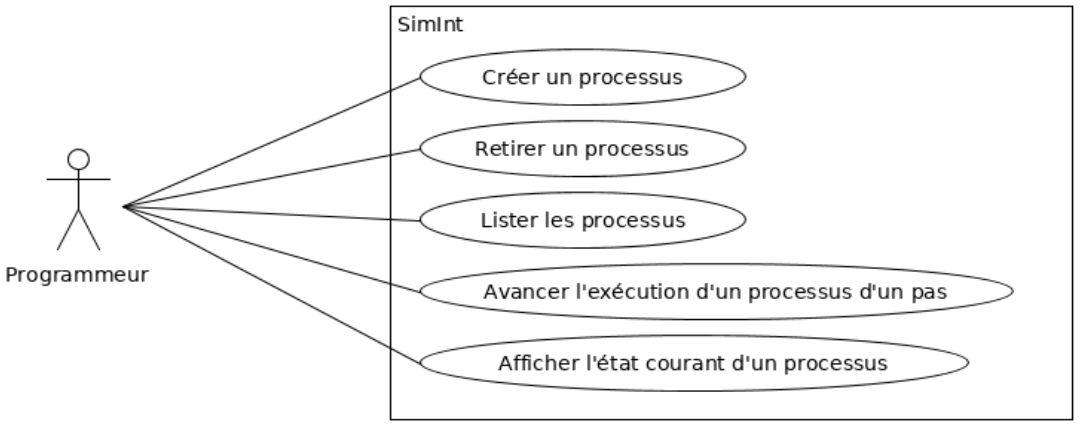
Table of Contents

1 Spécification

1.1 Diagrammes de cas d'utilisation







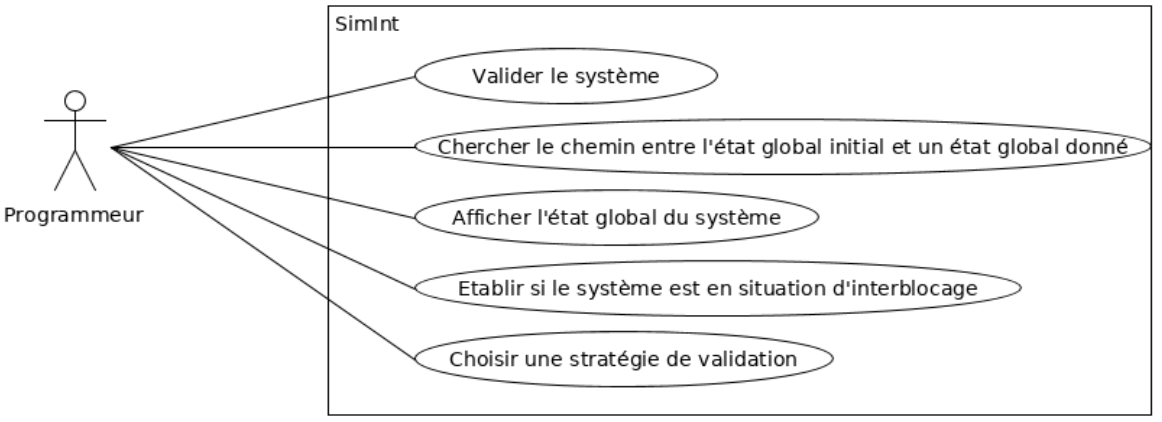


FIG. 1: Diagrammes de cas d'utilisation pour dans l'ordre: sémaphore, programme, processus, système

1.2 Priorités, préconditions et postconditions des cas d'utilisation

Les priorités des cas d'utilisation pour le sprint 1 sont choisies avec les règles de bon sens suivantes :

- pour retirer une entité du système, elle doit y être. La priorité de l'ajout est donc supérieure ou égale à la priorité du retrait ;
- pour lister les entités d'un type donné, elles doivent y être. La priorité de l'ajout est donc supérieure ou égale à la priorité du listage ;
- il est *a priori* possible, c.-à-d. sans raison contraire, de démontrer la mise en œuvre d'un sous-ensemble des fonctionnalités du système, et plus particulièrement la prise en compte des principales règles de gestion, sans les retraits ou les listages.
- la possibilité de lister aide au déverminage de l'application pendant les activités d'exécution des tests de validation.

Par conséquent, les cas d'utilisation d'ajout sont *a priori* de priorité « haute », ceux de listage de priorité « moyenne », et ceux de retrait de priorité « basse ».

Dans la suite, nous donnons les préconditions et postconditions pour les cas d'utilisation de priorité « Haute ». Pour les autres, nous indiquons uniquement leur niveau de priorité.

- Créer un sémaphore : HAUTE n°1
  - précondition : nom de sémaphore bien formé (non null ∧ non vide) ∧ sémaphore avec ce nom inexistant ∧ valeur initiale du compteur bien formée (supérieur ou égale à 0) ∧ exécution non débutée
  - postcondition : sémaphore initialisé avec ce nom existant
- Supprimer un sémaphore : basse
- Modifier l'état initial d'un sémaphore : basse
- Lister les sémaphores : moyenne
- Afficher l'état courant d'un sémaphore : moyenne
- Créer un programme : HAUTE n°2
  - précondition : nom de programme bien formé (non null ∧ non vide) ∧ programme avec ce nom inexistant ∧ exécution non débutée
  - postcondition : programme avec ce nom existant
- Supprimer un programme : basse
- Ajouter une instruction à un programme : HAUTE n°3
  - précondition :Type d'instruction non null ∧ nom sémaphore manipulé bien formé (non null et non vide) ∧ sémaphore manipulé avec ce nom existant ∧ nom programme bien formé (non null ∧ non vide) ∧ programme avec ce nom existant ∧ exécution non débutée

    - postcondition : instruction ajoutée au programme
- Retirer une instruction à un programme : basse
- Intervertir l'ordre de deux instructions d'un programme : basse
- Créer un processus : HAUTE n°4
    - précondition : nom de processus bien formé (non null ∧ non vide) ∧ processus avec ce nom inexistant ∧ exécution non débutée ^ nom programme bien formé (non null ∧ non vide) ∧ programme avec ce nom existant
    - postcondition : processus avec ce nom exécutant le programme
- Retirer un processus : basse
- Lister les processus : moyenne
- Avancer l'exécution d'un processus d'un pas : HAUTE n°5
    - précondition : nom processus bien formé (non null ∧ non vide) ∧ processus avec ce nom existant ∧ processus vivant
    - postcondition : exécution du processus avancée d'un pas
- Établir si le système est en situation d'interblocage : HAUTE n°6
    - précondition : exécution débutée
    - postcondition : vraie (pas de modification de l'état du système)
- Afficher l'état courant d'un processus : moyenne
- Afficher l'état global du système : moyenne
- Valider le système: HAUTE n°7
    - précondition : exécution non débutée
    - postcondition : exécution débutée
- Chercher le chemin entre l'état global initial et un état global donné : HAUTE n°8
    - précondition : état global atteignable
    - postcondition : chemin entre état global initial et état global donné
- Choisir une stratégie de validation : moyenne

2 Préparation des tests de validation

2.1 Tables de décision des tests de validation

La fiche programme du module CSC4102 ne permettant pas de développer des tests de validation couvrant l'ensemble des cas d'utilisation de l'application, les cas d'utilisation choisis sont de priorité HAUTE.

Numéro de test	1	2	3	4	5
Nom semaphore bien formé (≠ null∧ ≠ vide)	F	T	T	T	T
Semaphore inexistant avec ce nom		F	T	T	T
Valeur initiale du compteur bien formée (supérieur ou égale à 0)			F	T	T
Exécution non débutée				F	T
Création acceptée	F	F	F	F	T
Nombre de jeux de test	2	1	1	1	1

TAB. 1: Cas d'utilisation «créer un semaphore»

Numéro de test	1	2	3	4
Nom programme bien formé (≠ null∧ ≠ vide)	F	T	T	T
Programme inexistant avec ce nom		F	T	T
Exécution non débutée			F	T
Création acceptée	F	F	F	T
Nombre de jeux de test	2	1	1	1

TAB. 2: Cas d'utilisation «créer un programme»

Numéro de test	1	2	3	4	5	6	7
Type d'instruction ≠ null	F	T	T	T	T	T	T
Nom sémaphore manipulé bien formé (≠ null ∧ ≠ vide)		F	T	T	T	T	T
Sémaphore manipulé avec ce nom existant			F	T	T	T	T
Nom programme bien formé (≠ null ∧ ≠ vide)				F	T	T	T
Programme avec ce nom existant					F	T	T
Execution non débutée						F	T
Instruction ajoutée au programme		F	F	F	F	F	T
Nombre de jeux de test	2	1	2	1	2	1	1

TAB. 3: Cas d'utilisation «ajouter une instruction à un programme»

Numéro de test	1	2	3	4	5	6
Nom processus bien formé (≠ null∧ ≠ vide)	F	T	T	T	T	T
Exécution non débutée		F	T	T	T	T
Processus inexistant avec ce nom			F	T	T	T
Nom programme bien formé (≠ null∧ ≠ vide)				F	T	T
Programme existant avec ce nom					F	T
Création acceptée	F	F	F	F	F	T
Nombre de jeux de test	2	1	1	2	1	1

TAB. 4: Cas d'utilisation «créer un processus»

Numéro de test	1	2	3	4
Nom processus bien formé (≠ null∧ ≠ vide)	F	T	T	T
Processus avec ce nom existant		F	T	T
Processus vivant			F	T
Exécution du processus avancée d'un pas	F	F	F	T
Nombre de jeux de test	2	1	1	1

TAB. 5: Cas d'utilisation «avancer l'exécution d'un processus d'un pas»

Numéro de test	1	3
Exécution débutée	F	T
Situation d'interblocage ou non établie	F	T
Nombre de jeux de test	1	1

TAB. 6: Cas d'utilisation «établir si le système est en situation d'interblocage»

Numéro de test	1	3
Exécution non débutée	F	T
Exécution débutée	F	T
Nombre de jeux de test	1	1

TAB. 7: Cas d'utilisation «valider les sytème»

Numéro de test	1	3
État global donné atteignable	F	T
Chemin entre l'état global initial et l'état global donné	F	T
Nombre de jeux de test	1	1

TAB. 8: Cas d'utilisation «chercher le chemin entre l'état global initial et un état global donné»

3 Conception

3.1 Liste des classes

À la suite d'un parcours des diagrammes de cas d'utilisation et d'une relecture de l'étude de cas, voici une première liste de classes avec quelques attributs :

- SimInt (la façade)
- Processus — nom
- ÉtatGlobal (l'état du système) — , etatExecution
- ÉtatProcessus (la partie de l'état concernant les processus) — etat, instructionCourante
- Sémaphore – nom, valeurInitiale
- EtatSémaphore – valeurCompteur
- Programme – nom
- Instruction – typeInstruction
- ModelChecker

3.2 Diagramme de classes

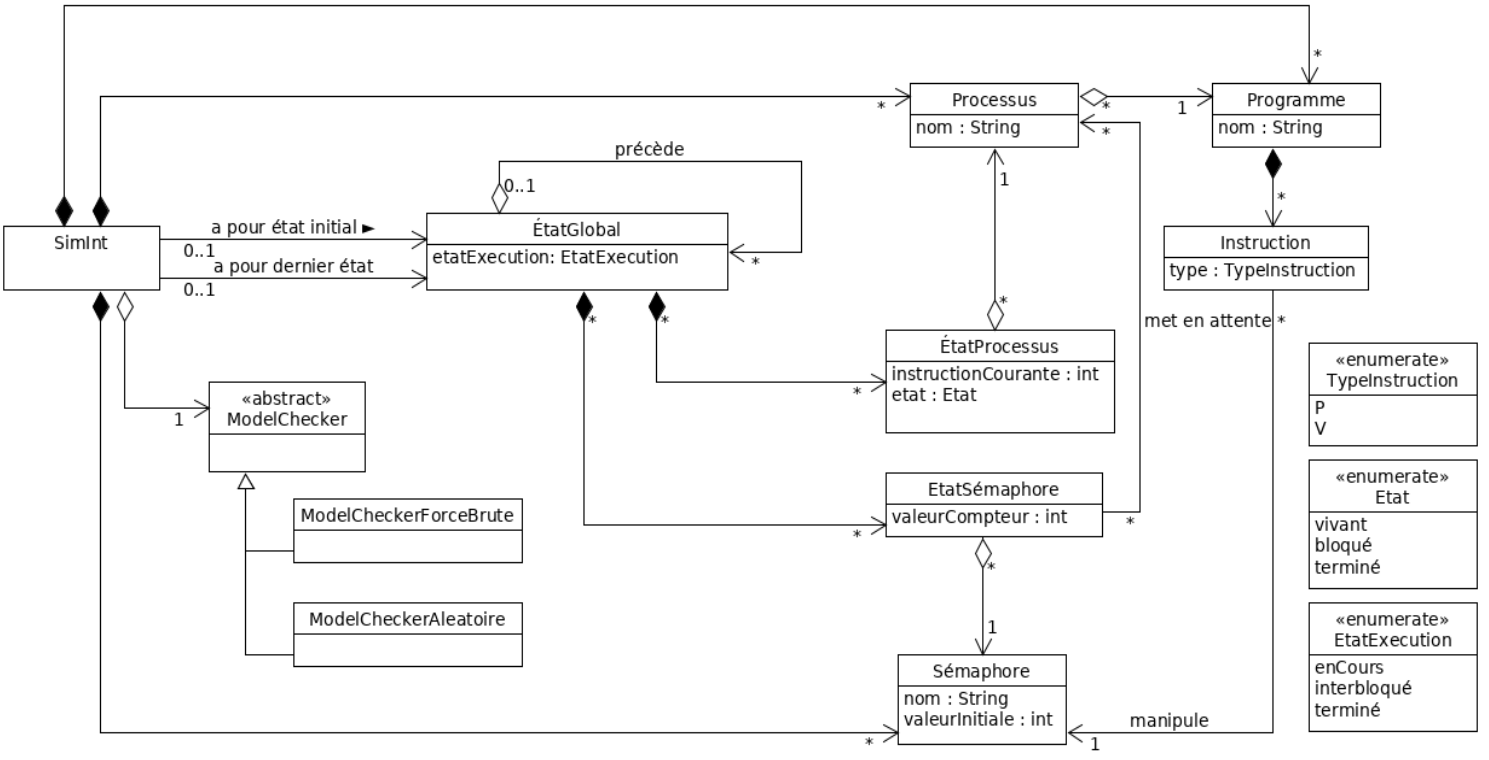


FIG. 1: Diagramme de classes

3.3 Diagrammes de séquence

Voici la description textuelle du cas d'utilisation « créer un semaphore» :

- arguments en entrée : nom du semaphore, valeur initiale du compteur
- rappel de la précondition : nom de semaphore bien formé (non null ∧ non vide) ∧ semaphore avec ce nom inexistant ∧ valeur initiale du compteur bien formée (supérieur ou égale à 0) ∧ exécution non débutée
- algorithme :
    1. vérifier les arguments
    2. si en outre l'exécution du système n'a pas débuté
        (a) vérifier que le semaphore n'existe pas
        (b) créer un sémaphore en initialisant son compteur avec la valeur initiale et avec son nom et ajouter à la collection de sémaphore
        (e) ajouter un état dans l'état global initial pour ce sémapore
            i. créer un état pour ce sémphore
            ii. ajouter l'état sémaphore à l'état global initial

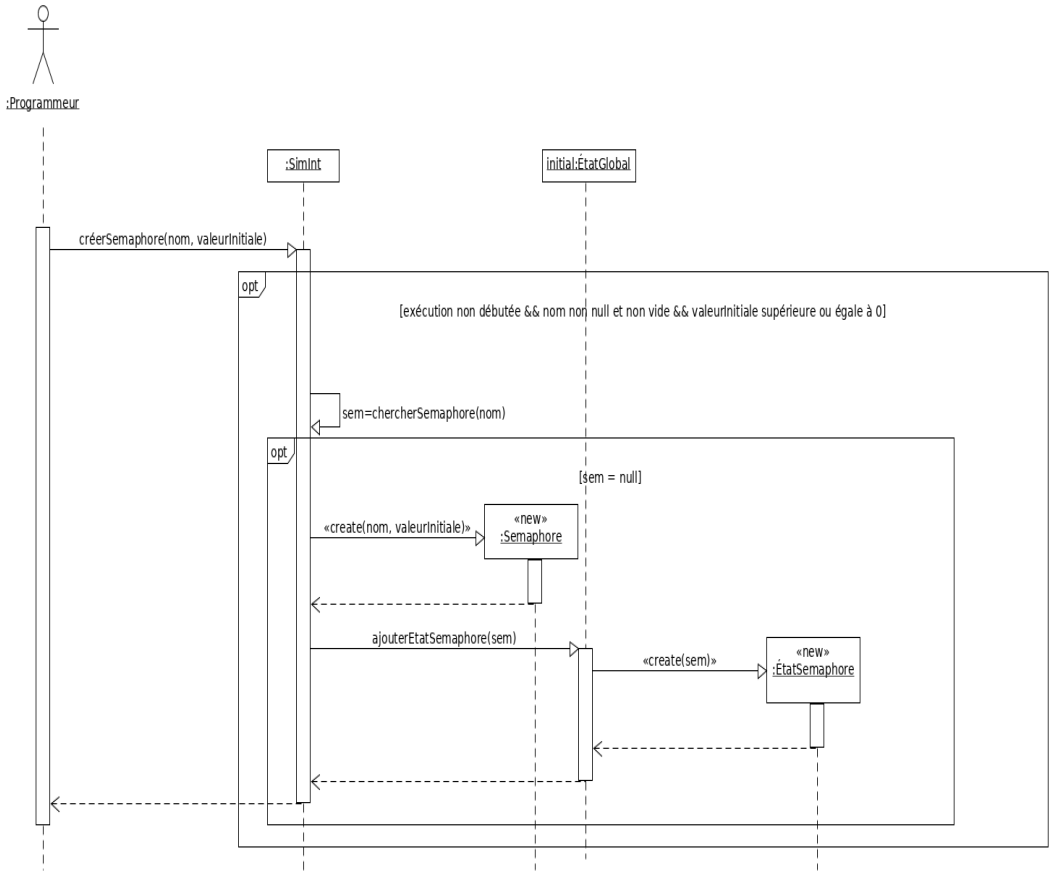


FIG. 4: Diagramme de séquence du cas d'utilisation «créer un semaphore»

Voici la description textuelle du cas d'utilisation « créer un programme» :

- arguments en entrée : nom du programme
- rappel de la précondition : nom de programme bien formé (non null ∧ non vide) ∧ programme avec ce nom inexistant ∧ exécution non débutée
- algorithme :
    1. vérifier les arguments
    2. si en outre l'exécution du système n'a pas débuté
        (a) vérifier que le programme n'existe pas
        (b) créer un programme avec son nom et ajouter à la collection de progammes

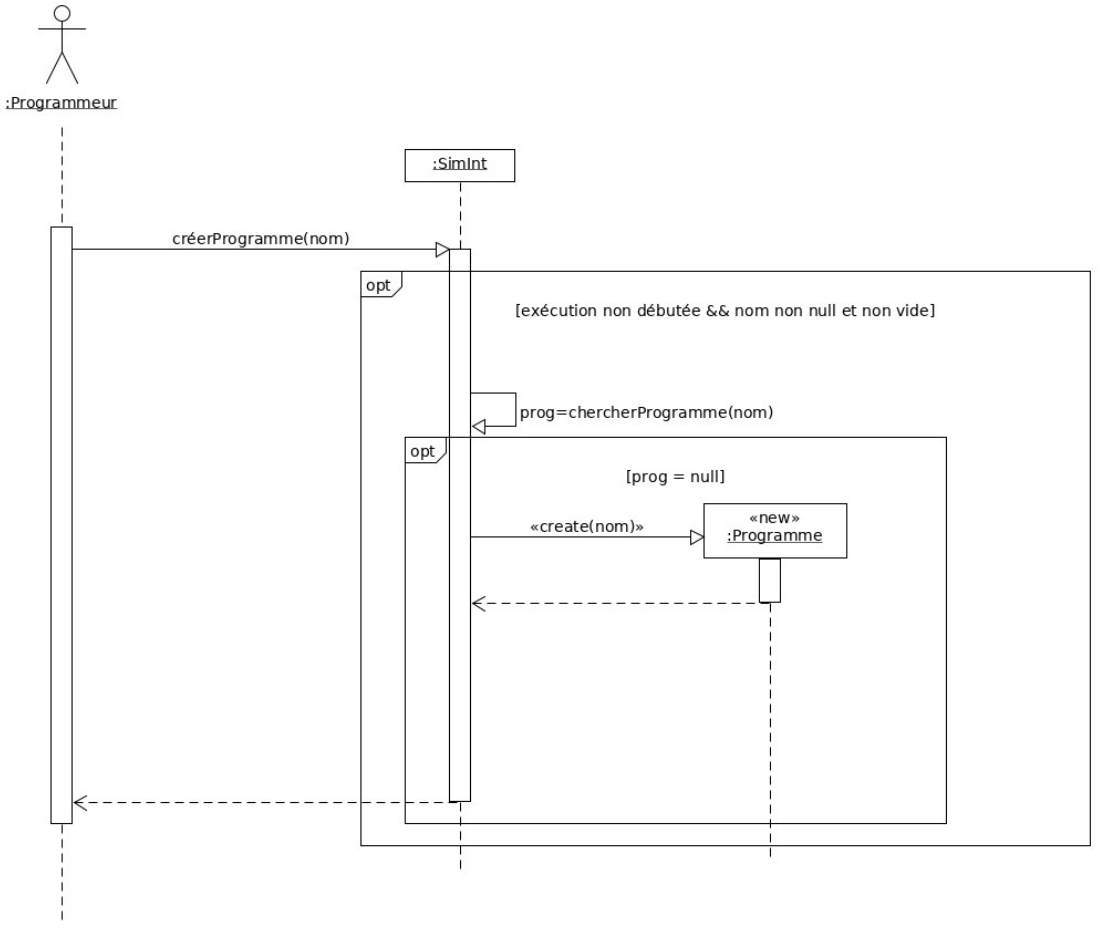


FIG. 5: Diagramme de séquence du cas d'utilisation «créer un programme»

Voici la description textuelle du cas d'utilisation « ajouter une instruction à un programme » :

– arguments en entrée : typeInstruction, nom du semaphore, nom du programme

– rappel de la précondition : typeInstruction non null ∧ nom semaphore manipulé bien formé (non null et non vide) ∧ semaphore manipulé avec ce nom existant ∧ nom programme bien formé (non null ∧ non vide) ∧ programme avec ce nom existant ∧ exécution non debutée

– algorithme :

1. vérifier les arguments et que l'exécution n'a pas débutée
2. si le semaphore existe et le programme existe

   a) compter le nombre d'instructions du programme

   b) ajouter l'instruction au programme en fonction de son type avec comme numéro d'instruction : nombre d'instructions + 1

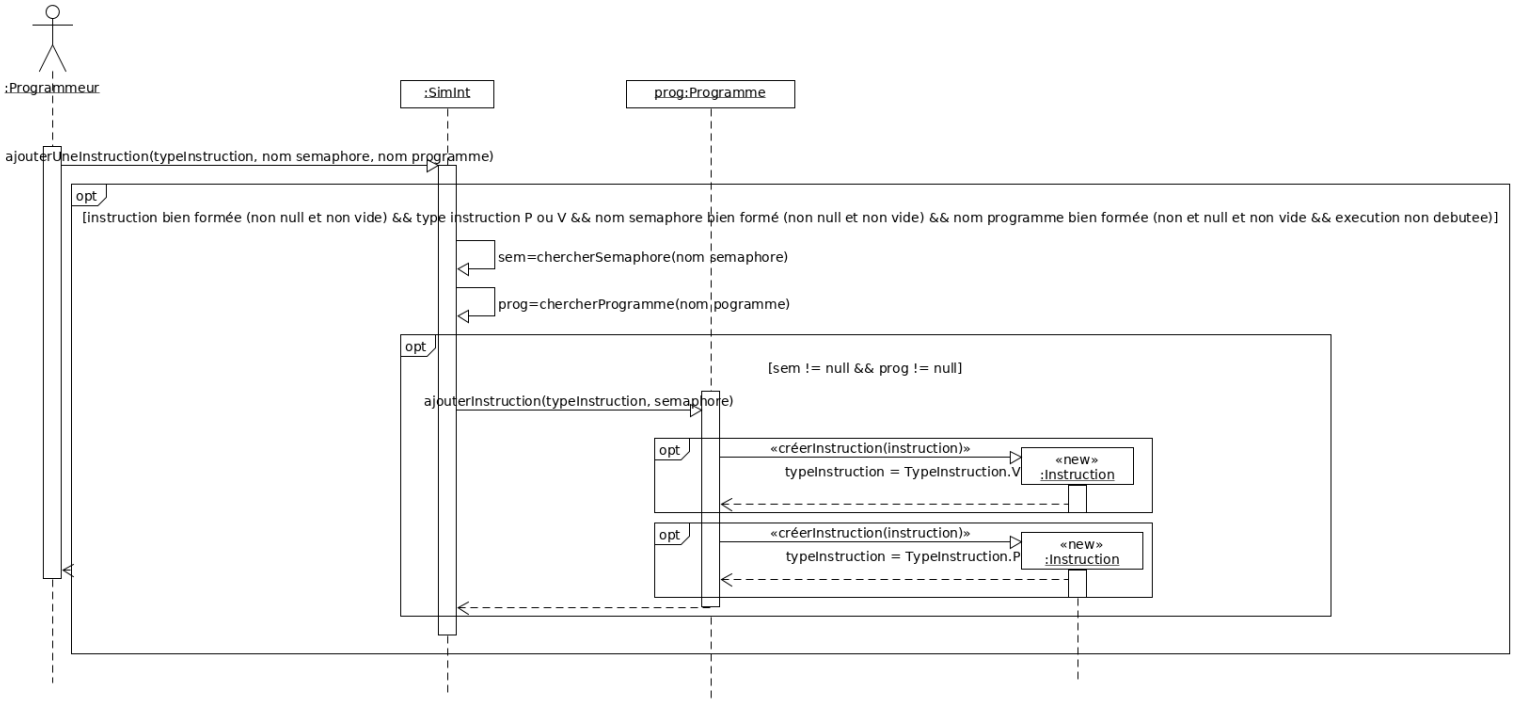


FIG. 6: Diagramme de séquence du cas d'utilisation « ajouter une instruction à un programme»

Voici la description textuelle du cas d'utilisation « créer un processus » :

– arguments en entrée : nom du processus, nom du programme

– rappel de la précondition : nom de processus bien formé (non null ∧ non vide) ∧ processus avec ce nom inexistant ∧ exécution non débutée ^ nom programme bien formé (non null ∧ non vide) ∧ programme avec ce nom existant

– algorithme :

1. vérifier les arguments
2. si en outre l'exécution du système n'a pas débuté

   (a) vérifier que le processus n'existe pas

   (b) vérifier que le programme existe

   (c) créer un processus avec ce nom et la référence du programme

   (d) ajouter le processus à la collection des processus

   (e) ajouter un état dans l'état global initial pour ce processus

      i. créer un état pour ce processus

      ii. ajouter l'état de processus à l'état global initial

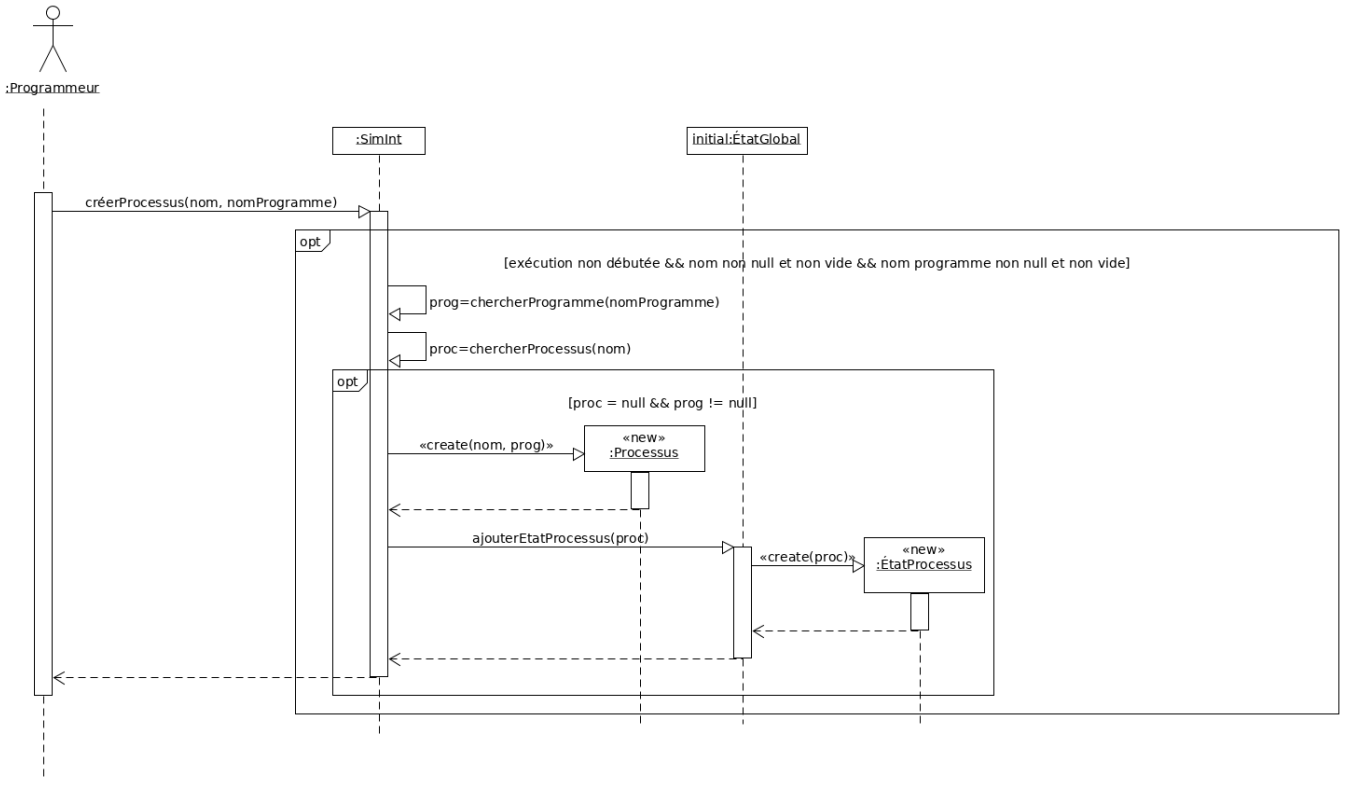


FIG. 7: Diagramme de séquence du cas d'utilisation «créer un processus»

Voici la description textuelle du cas d'utilisation « avancer l'exécution d'un processus d'un pas » :

– arguments en entrée : nom processus

– rappel de la précondition : nom processus bien formé (non null ∧ non vide) ∧ processus avec ce nom existant ∧ etat vivant

– algorithme :

1. vérifier les arguments
2. si le processus existe, cherche l'état de ce processus
3. si le processus n'est pas terminé
   a) copie de l'état courant vers un nouvel état
   b) cherche l'état du processus copié
   c) cherche l'instruction du programme à exécuter par le processus
   d) exécute cette instruction

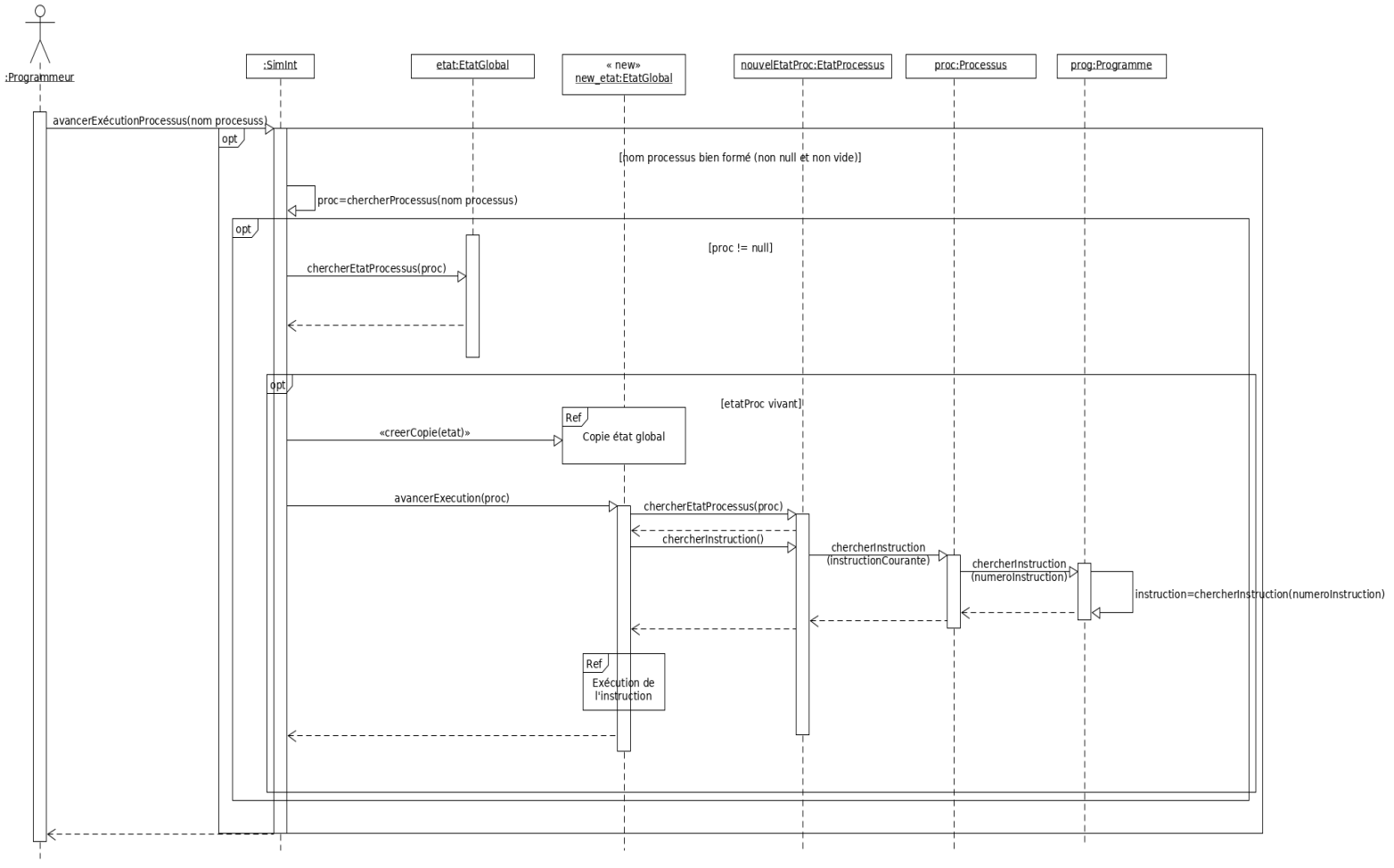


FIG. 8: Diagramme de séquence du cas d'utilisation «avancer l'exécution d'un processus d'un pas»

Voici la description textuelle du cas d'utilisation « établir si le système est en situation d'interblocage » :

– rappel de la précondition : exécution débutée

– algorithme :

1. vérifier que l'exécution est débutée

2. établir si le système est en interblocage

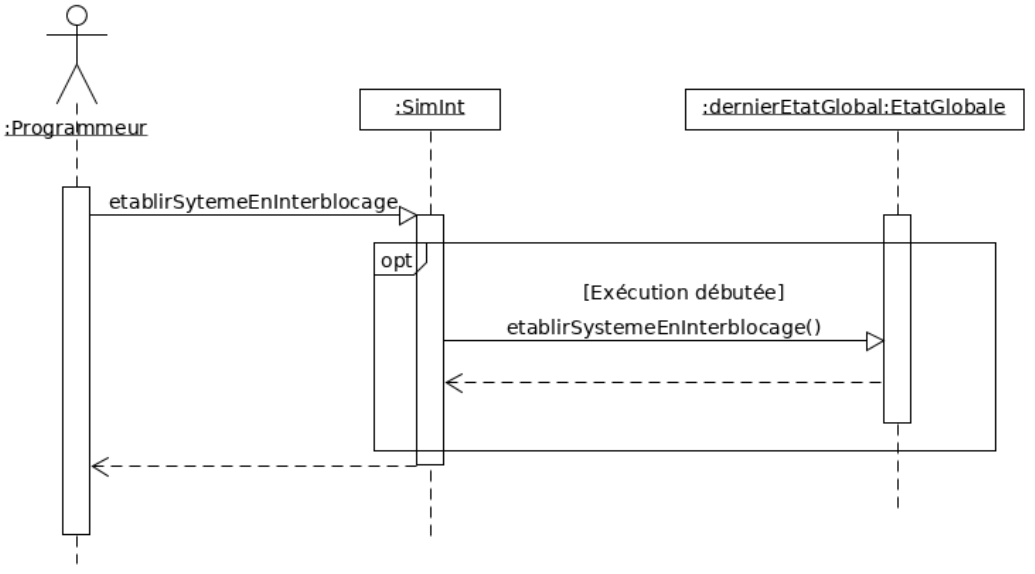


FIG. 9: Diagramme de séquence du cas d'utilisation «établir si le système est en situation d'interblocage»

Voici la description textuelle du cas d'utilisation « valider le système :

- rappel de la précondition : exécution non débutée
- algorithme :
    1. Vérifier que l'exécution est non débutée
    2. débuter l'exécution
    3. valider le système

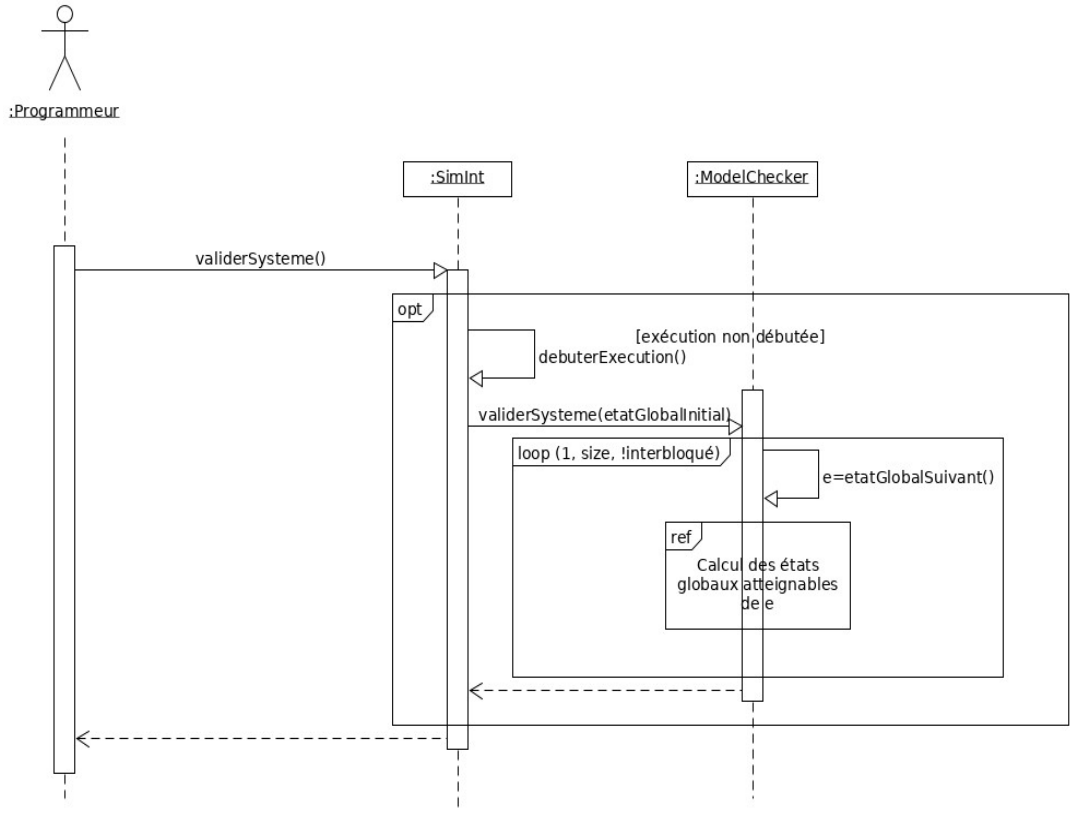


FIG. 10: Diagramme de séquence du cas d'utilisation «Valider le système»

4 Fiche des classes

4.1 Classe SimInt

Simint
<– attributs « association » –> − processus : collection de @Processus − semaphores : collection de @Semaphore − programmes : collection de @Programme − étatGlobalInitial : @ÉtatGlobal − dernierEtatGlobal : @EtatGlobal − exécutionDébutée : booléen
– modelChecker : @ModelChecker
<– constructeur –> + SimInt() <– operations « cas d'utilisation » –> + créerProcessus(String nom, String nomProg) : void + chercherProcessus(String nom) : @Processus + créerSemaphore(String nom, Integer valeurInitiale) : void + chercherSemaphore(String nom) : @Semaphore + créerProgramme(String nom) : void + chercherProgramme(String nom) : @Programme + établirSystemeInterblocage() : @EtatExecution + validerSysteme() : boolean + ajouterInstruction(String nomProg, String nomSem, TypeInstruction type) : void + avancerExecutionProcessus(String nomProcessus) : @EtatGlobal + chercherChemin(@EtatGlobal etatGlobal) : void + setModelChecker() : void <– operations « nouveau cas d'utilisation »–> + afficherEtatsProcessus() : void + afficherEtatsSemaphore() : void + listerProcessus() : void + listerSemaphore() : void + listerProgramme() : void + supprimerProcessus(String nom) : void + supprimerProgramme(String nom) : void + supprimerSemaphore(String nom) : void - existeProcessus(String nom) : void - existeProgramme(String nom) : void - existeSemaphore(String nom) : void

4.2 Classe Processus

Processus
<– attributs –> − nom : String <− attributs « associations » −> − programme : @Programme
<– constructeur –> + constructeurProcessus(String nom, String nomProgramme) <− opérations « cas d'utilisation » −> + chercherInstruction(Integer numeroInstruction) : @Instruction

4.3 Classe ÉtatProcessus

ÉtatProcessus
<– attributs –> − processus : @Processus – etat : Etat=Etat.vivant – instructionCourante : integer − compteurInstanciation : integer = 0 − compteurInstance : integer <− attributs « associations » −> – processus : @Processus
<– constructeur –> + constructeurEtatProcessus(@Processus processus) + constructeurEtatProcessus(@EtatProcessus origine) <– opérations « cas d'utilisation » –> + avancerExecution() : void + chercherInstruction() : @Instruction + chaineDeCaracteres() : String <– opérations « nouveau cas d'utilisation » –> + afficherEtatProcessus() : void

4.3 Classe Semaphore

Semaphore
<– attributs –> − nom : String − valeurInitiale : integer
<– constructeur –> + constructeurProcessus(String nom, Integer valeurInitiale)

4.4 Classe ÉtatSemaphore

ÉtatSemaphore
<– attributs –> – valeurCompteur : integer – compteurInstanciation : integer = 0 – compteurInstance : integer <– attributs « associations » –> – fileAttente : Collection de @Processus – semaphore : @Semaphore
<– constructeur –> + constructeurSemaphore(@Semaphore semaphore) + constructeurSemaphore(@EtatSemaphore origine) <– opérations « cas d'utilisation » –> + mettreProcessusEnAttente(@Processus processus) : void + libérerProcessus() : @Processus + chaineDeCaracteres() : String <– opérations « nouveau cas d'utilisation » –> + afficherEtatSemaphore() : void

4.5 Classe Programme

Programme
<– attributs –> − nom : String <− attributs « associations » −> − instructions : Collection de @Instruction
<– constructeur –> + constructeurProgramme(String nom) + destructeur() <− opérations « cas d'utilisation » −> + ajouterInstruction(@TypeInstruction type, @Semaphore sem) : void + chercherInstruction(Integer numeroInstruction) : @Instruction <− opérations « nouveau cas d'utilisation » −> + supprimerInstruction(Integer numeroInstruction) : void + modifierOrdreInstruction(Instruction instruction1, Instruction instruction2) : void

4.6 Classe Instruction

Instruction
<– attributs –> − typeInstruction : @TypeInstruction <− attributs « associations » −> − semaphore : @Semaphore
<– constructeur –> + constructeurProcessus(@TypeInstruction type, @Semaphore semaphore)

4.7 Classe ÉtatGlobal

ÉtatGlobal
<– attributs –> − etatGlobalPrecedent : @EtatGlobal − étatsGlobauxAtteignables : collection de @ÉtatGlobal − etatExecution : @EtatExecution − compteurInstanciation : integer = 0 − compteurInstance : integer - chaineDeCaracteres : string <– attributs « association » –> − étatsProcessus : collection ordonnée de @ÉtatProcessus − étatsSemaphore : collection ordonnée de @ÉtatSemaphore
<– constructeurs –> + constructeurÉtatGlobal() + constructeurÉtatGlobal(@EtatGlobal origine) // constructeur par copie <– operations « cas d'utilisation » –> + avancerExecution(String nomProcessus) : void + ajouterÉtatProcessus(@Processus processus) : boolean + chercherÉtatProcessus(String nom) : @ÉtatProcessus + ajouterEtatSemaphore(@Semaphore semaphore) : void + chercherEtatSemaphore(String nom) : @EtatSemaphore + etablirSystèmeEnInterbloquage() : void + ajouterEtatGloablAtteignable(@Etatglobal etatGlobal) : void - computeChaineDeCaracteres() : String <– operations « nouveau cas d'utilisation » –> + afficherEtatsProcessus() : void + afficherEtatsSemaphore() : void + listerEtatsProcessus() : void + listerEtatsSemaphore() : void

5 Diagrammes de machine à états et invariants

5.1 Classes Processus

L'invariant de la classe Processus est le suivant :

*nom≠null∧nom≠“"∧*programme≠null

5.2 Classes ÉtatProcessus

L'invariant de la classe ÉtatProcessus est le suivant :

processus≠null∧instructionCourante>=0∧etat≠null∧compteurInstance>0∧compteurInstanciation>0

5.3 Classes ÉtatGlobal

L'invariant de la classe ÉtatGlobal est le suivant :

etatsProcessus≠null∧etatsSemaphore≠null∧etatExecution≠null∧compteurInstance>0∧compteurInstanciation>0∧processus bloqué dans une file d'attente

5.4 Classes ÉtatSemaphore

L'invariant de la classe ÉtatSemaphore est le suivant :

semaphore≠null∧valeurCompteur>=0∧compteurInstance>0∧compteurInstanciation>0

5.5 Classes Semaphore

L'invariant de la classe Semaphore est le suivant :

nom≠null∧nom≠“"∧valeurInitiale>=0

5.6 Classes Programme

L'invariant de la classe Programme est le suivant :

nom≠null∧nom≠“"

5.7 Classes Instruction

L'invariant de la classe Instruction est le suivant :

type≠null∧ semaphore≠null

6 Préparation des tests unitaires

6.1 Classe Processus

Numéro de test	1	2	3
nom ≠*null*∧≠*vide*	F	T	T
p*rogramme ≠ null*		F	T
nom'=nom			T
programme'=programme			T
invariant			T
Levée d'une exception	OUI	OUI	NON
Objet créé	F	F	T
Nombre de jeux de test	2	1	1

TAB. 2: Méthode `constructeurProcessus` de la classe `Processus`»

6.2 Classe EtatProcessus

Numéro de test	1	2
p*rocessus ≠ null*	F	T
processus'=*processus*		T
compteurInstanciation'=compteurInstanciation+1		T
compteurInstance'=compteurInstanciation'		T
invariant		T
Levée d'une exception	OUI	NON
Objet créé	F	T
Nombre de jeux de test	1	1

TAB. 3: Méthode `constructeurEtatProcessus` de la classe Etat`Processus`»

Numéro de test	1	2
origine ≠ *null*	F	T
processus'=*origine.processus*		T
compteurInstanciation'=compteurInstanciation+1		T
compteurInstance'=compteurInstanciation'		T
invariant		T
Levée d'une exception	OUI	NON
Objet créé	F	T
Nombre de jeux de test	1	1

TAB. 4: Méthode `constructeurParCopieEtatProcessus` de la classe Etat`Processus`»

Numéro de test	1	2
etat=Etat.vivant	F	T
invariant		T
Levée d'une exception	OUI	NON
Objet créé	F	T
Nombre de jeux de test	1	1

TAB. 5: Méthode `avancerExecution` de la classe Etat`Processus`»

6.3 Classe Semaphore

Numéro de test	1	2	3
`nom` *≠null∧≠vide*	F	T	T
valeurInitiale>=0		F	T
nom'=nom			T
valeurInitiale'=valeurInitiale			T
invariant			T
Levée d'une exception	OUI	OUI	NON
Objet créé	F	F	T
Nombre de jeux de test	2	1	1

TAB. 6: Méthode `constructeurSemaphore` de la classe `Semaphore`»

6.4 Classe EtatSemaphore

Numéro de test	1	2
`semaphore` *≠null*	F	T
semaphore'=semaphore		T
valeurCompteur'=semaphore.valeurInitiale		T
compteurInstanciation'=compteurInstanciation+1		T
compteurInstance'=compteurInstanciation'		T
invariant		T
Levée d'une exception	OUI	NON
Objet créé	F	T
Nombre de jeux de test	1	1

TAB. 7: Méthode `constructeurEtatSemaphore` de la classe Etat`Semaphore`»

Numéro de test	1	6
`origine` ≠*null*	F	T
semaphore'=origine.semaphore		T
valeurCompteur'=origine.valeurCompteur		T
compteurInstanciation'=compteurInstanciation+1		T
compteurInstance'=compteurInstanciation'		T
invariant		T
Levée d'une exception	OUI	NON
Objet créé	F	T
Nombre de jeux de test	1	1

TAB. 8: Méthode `constructeurParCopieEtatSemaphore` de la classe Etat`Semaphore`»

6.5 Classe Programme

Numéro de test	1	2
`nom` ≠*null*∧≠*vide*	F	T
nom'=nom		T
invariant		T
Levée d'une exception	OUI	NON
Objet créé	F	T
Nombre de jeux de test	2	1

TAB. 9: Méthode `constructeurProgramme` de la classe `programme`»

6.6 Classe Instruction

Numéro de test	1	2	3
type≠*null*	F	T	T
sem≠*null*		F	T
type'=type			T
sem'≠*null*			T
invariant			T
Levée d'une exception	OUI	OUI	NON
Objet créé	F	F	T
Nombre de jeux de test	1	1	1

TAB. 10: Méthode `constructeurInstruction` de la classe `Instruction`»

6.7 Classe EtatGlobal

Numéro de test	1
etatsProcessus'≠null	T
etatsSemaphore'≠null	T
etatExecution'≠null	T
etatGlobalPrecedent' = null	T
compteurInstanciation'=compteurInstanciation+1	T
compteurInstance'=compteurInstanciation'	T
invariant	T
Levée d'une exception	NON
Objet créé	T
Nombre de jeux de test	1

TAB. 11: Méthode `constructeurEtatGlobal` *de la classe* `EtatGlobal`*»*

Numéro de test	1	7
origine*≠null*	F	T
etatsProcessus'≠null		T
etatsSemaphore'≠null		T
etatExecution'≠null		T
etatGlobalPrecedent' = origine		T
compteurInstanciation'=compteurInstanciation+1		T
compteurInstance'=compteurInstanciation'		T
invariant		T
Levée d'une exception	OUI	NON
Objet créé	F	T
Nombre de jeux de test	1	1

TAB. 12: Méthode `constructeurParCopieEtatGlobal` *de la classe* `EtatGlobal`*»*