# Analyse et modélisation d'un simulateur multi-agents basé sur la théorie constructale

Version 1.0

décembre 2005

Rédigé par

**Jean-Christophe Denaës**

dans le cadre du cours DIC9250 de Hakim Lounis

**UQAM**

# Sommaire

# Illustrations

# 1. Introduction

Le présent projet consiste à réaliser un simulateur dans le but d'expérimenter un nouveau type de système multi-agents réactifs. Le logiciel résultant devra permettre aux chercheurs intéressés de découvrir et d'explorer une conception constructale de l'intelligence et de la vie (artificielles). Le logiciel de simulation devra autoriser, par exemple, la navigation autonome d'un robot. De plus, il sera augmenté par une interface de gestion et de visualisation des simulations.

Ce document décrit essentiellement l'interface de gestion du simulateur et sommairement le comportement du SMA. Il ne traite pas de l'algorithme dérivé de la Théorie constructale d'Adrian Bejan.

## *1.1 Objectifs*

L'objectif du document est de décrire les spécifications du logiciel en précisant de manière descriptive un ensemble d'éléments répondant aux exigences d'un système logiciel. Ce document décrit les fonctionnalités du futur système et, bien que conservant une certaine profondeur de détails, les abordera avec une certaine abstraction afin d'obtenir une vue générale simplifiée du dit logiciel. Celui-ci, bien qu'utilisable par des chercheurs de différents domaines, vise avant tout les philosophes de la cognition qui sont intéressés à avoir des outils informatiques plus spécifiques à leur champ d'étude.

## *1.2 Portée*

Le simulateur, dénommé ***Ka***[1], s'occupera de générer un environnement dynamique et évolutif par l'entremise d'un SMA-R et l'usage d'un algorithme conçu à cet effet.

Il est attendu que le logiciel puisse simuler le comportement de systèmes dynamiques à réaction-diffusion, oscillants et auto-oscillants, et qu'il puisse être implanté dans un logiciel de visualisation et d'analyse ou couplé à d'autre composants, physiques ou logiques, comme des réseaux de neurones ou un robot.

Il n'est pas attendu que ce logiciel soit équivalent, dans sa finalité utilitaire, aux logiciels

1 Dans la religion égyptienne, *Ka* est le mot désignant l'ensemble des énergies vitales animant les dieux et les hommes.

d'intelligence artificielle. Par exemple, il n'est pas attendu de faire de la reconnaissance visuelle ou sonore avec ce seul simulateur, mais plutôt qu'il améliore significativement les performances de divers réseaux neuronaux. Aussi, il est attendu de ce logiciel qu'il rende compte de son environnement de manière signifiante, et que cette représentation soit utiliser pour le guidage robotique ou encore la reconnaissance sonore.

Ce logiciel est un prototype qui se veut avant tout être un outil pour les philosophes utilisant la simulation dans leurs travaux. De ce point de vue, l'objectif est d'abord de créer un simulateur spécifiquement construit pour être utilisé en philosophie de la cognition. Le retombées seront significatives quand aux questions philosophiques portant sur la simulation ou encore le problème de l'homunculus-sub-homunculi; questions dont les réponses orienteront sans nul doute les futurs travaux en IA, VA et informatique cognitive.  Précisément, ce simulateur souhaite permettre d'étudier la possible conception relationnelle (ondulatoire ou « liquide ») du vivant et donc de la cognition.

## *1.3 Définitions, acronymes et abréviations*

- CORBA Common object request broker architecture
- DEVS Discrete-Events Simulation
- HLSIM High Level Simulator (for Gunk)
- IA Intelligence artificielle
- LSM Liquid State Machines
- MGS Modèle Général de Simulation (de systèmes dynamiques)
- UI User Interface (interface utilisateur)
- SD Système dynamique
- SMA-R Système multi-agents réactifs
- TC Théorie constructale
- UML Unified Modelling Language
- VA Vie artificielle

## *1.4 Documents de références*

Roques, Pascal. *UML par la pratique : Cours et exercices Java et C++*. Paris : Eyrolles; 2003.

Larman, Craig. *Applying UML and patterns : an introduction to object-oriented analysis and design and the unified process*. 2nd ed. Upper Saddle River, NJ : Prentice Hall PTR; 2002.

Blaha, Michael & Rumbaugh, James. *Modélisation et conception orientées objet avec UML 2.* 2nd ed. Pearson Education; 2005.

Gamma, Erich. *Design patterns elements of reusable object-oriented software*. Reading, Mass. ; Don Mills, Ont: Addison-Wesley; 1995

D. Zinoviev, *Mapping DEVS Models onto UML Models*, Proc. of the 2005 DEVS Integrative M&S Symposium, San Diego, CA, April 2005, pp. 101-106 http://arxiv.org/PS_cache/cs/pdf/0508/0508128.pdf

## *1.5 Aperçu du document*

Ce document décrit les fonctionnalités du logiciel. Il est divisé en quatre sections.

- La première section est celle de l'introduction qui donne une brève description du logiciel.
- La seconde section est la description générale du logiciel qui vise à donner une vue d'ensemble du simulateur *Ka*.
- La troisième est celle de la description détaillée. Elle permet de décrire les exigences du système de manière suffisamment précise pour entrevoir la conception de ce dernier. C'est aussi dans cette partie que se trouve le dictionnaire de données.
- La dernière section est celle des informations complémentaires et qui comporte, au besoin, un index et des annexes.

# 2. Description générale du logiciel

## *2.1 Perspective du produit*

Présentement, l'ensemble des logiciels et des algorithmes que les simulateurs en IA et VA mettent en oeuvre sont basés sur la vision dominante de notre époque et qui tient toute entière dans la notion d'*objet* (la modélisation *orienté objet* est en ce sens une extension de cette perception *objet* que nous avons de notre environnement). Le concept sous-jacent d'une telle approche est la relation "point à point" qui existe entre les objets (relation d'objet à objet). Aussi, les simulateurs actuels peuvent être considérés comme des outils analytiques plus ou moins sophistiqués, i.e. construits autour de grammaires formelles (dont les grammaires génératives)  celle de Lindenmayer (*L-System*) et, plus récemment, de Păun (*P-System*).

Les simulateurs qui semblent commencer à se démarquer et à intégrer des modèles de réaction-diffusion multi-échelles sont **E-Cell** (http://www.e-cell.org/about/what) et **Cell-DEVS** (http://www.sce.carleton.ca/faculty/wainer/celldevs). Tous deux sont développés autour du formalisme DEVS. (http://www.acims.arizona.edu). D'autres simulateurs peuvent aussi être nommée dans le domaine de l'informatique amorphe inspirée par les sytèmes chimiques, comme **HLSIM** du MIT (http://www.swiss.csail.mit.edu/projects/amorphous) et **MGS** du LaMI (http://mgs.lami.univ-evry.fr) mais ils restent globalement analytiques au sens précité.

Le simulateur possédera les spécifications d'interfaces suivantes:

- En soi, le logiciel (module de simulation) n'aura qu'une interface de communication minimale qui pourra être coupler à celle d'un robot ou à une interface de communication plus élaborée (cf. point suivant).
- Il devra aussi y avoir une interface usagée (logiciel satellite au premier) permettant de paramétrer le module de simulation, de charger ou sauvegarder des modèles pré-paramétrés, mais aussi de visualiser les simulations et de générer des rapports sur ces dernières.
- Le choix du ou des langages de développement n'est pas encore arrêté, mais l'on peut d'ores et déjà avancé qu'il s'agira d'une combinaison entre un langage symbolique (LISP) et un langage objet (C++, Java). Fortran est aussi un langage

de programmation toujours fortement utilisé dans le domaine de recherche investigué.

Au niveau de l'interface utilisateur, un outil très visuel sera utilisé (certainement Matlab) afin d'améliorer l'accessibilité et la convivialité de l'application. Cette interface donnera accès au module de simulation qu'il soit implanter localement ou `distance (e.g. dans un robot). Elle permettra aussi l'émulation d'un robot virtuel (miniBot) pour les expérimentation.

## *2.2 Vue d'ensemble des fonctions du produit*

Nous proposons ici une liste exhaustive des principales fonctions du système sous forme de module (ici, un module est un logiciel à par entière).

| **Module:** Simulateur |
|---|

Ce module englobe tous les processus relatifs à la génération d'une simulation.

- Initialiser et terminer une simulation via un gestionnaire d'évènements (connecté à un interrupteur manuel ou un minuteur).
- Simuler un système à réaction-diffusion.

| **Module:** Interface Utilisateur |
|---|

- Gestion des paramètres
    1. Charger une liste de paramètres.
    2. Générer une liste de paramètres.
    3. Modifier  une liste de paramètres.
    4. Sauvegarder une liste de paramètres.

- Gestion d'une simulation
    1. Visualiser une simulation (données brute, 2D, 3D).
    2. Générer un rapport d'activités.

## *2.3 Description des utilisateurs*

Pour ce système, nous considérerons deux classes d'utilisateurs, soit le robot et l'utilisateur humain (le chercheur).

| **Utilisateur:** le robot |
|---|

Il est l'utilisateur de base, celui qui initialise le simulateur et qui exploite la simulation pour, par exemple, se déplacer.

- Tâches:
  - Initialiser le simulateur.
  - Utiliser la simulation (à l'aide d'autres composants informatique et électronique).

- Niveau d'autorité
  - Opérationnel

| **Utilisateur:** le chercheur |
|---|

Génériquement nommée « *user* », il est celui qui manipulera et exploitera le plus le système en émulant un robot virtuel, mais surtout en analysant et exploitant les données fournies par les simulations qu'il aura généré.

- Tâches:
  - Initialiser le simulateur.
  - Utiliser la simulation (à l'aide d'autres composants informatique et électronique).
  - Suivre (visualiser) la simulation (en temps réel ou différé).
  - Créer de nouvelles simulations.
  - Analyser et exploiter les rapports de simulation.

- Niveau d'autorité
  - Opérationnel
  - Décisionnel

## *2.4 Contraintes d'ordre général*

Les contraintes ne sont pas à l'heure actuelles véritablement évaluée car le projet reste techniquement non précisé.

On peut toutefois déjà dire que :

- Le développement de l'algorithme du simulateur devra se rapprocher de standard comme DEVS (et CORBA).
- Le simulateur devra pouvoir être interfacé avec des composants électroniques et informatiques.
- Le simulateur devra être développé en tenant compte d'une évolution possible vers le des architectures parallèles et distribuées.
- Les langages de programmation devront donc être choisi selon trois critères : Rapidité de calcul à l'exécution (LISP), capacité à intégrer des solutions autour du parallélismes et des grilles de calcul (C++, Python, Fortran, Java).
- Les paramètres (dont les séquences d'événements) provenant de sources externes et étant soumis à des contraintes algorithmiques, un format devra être respecté.

## *2.5 Hypothèses et dépendances*

Prémisses de base qui pourraient affecter les exigences:
- Nombre et expérience des développeurs.
- Ressources requises par l'algorithme (actuellement non évalué)
- Obtention du matériel requis pour le développement.

### *2.6 Répartition des exigences*

Répartition définie d'après les cas d'utilisations énumérés.

| **Priorité** | **Fonctionnalité** |
|---|---|
| Haute | La gestion du simulateur (DEVS + algorithme constructal) |
| Haute | La gestion de l'activation et de l'arrêt de la simulation |
| Moyenne | La gestion de la visualisation de la simulation |
| Basse | La gestion des paramètres (via une interface utilisateur) |
| Basse | La gestion de la génération de rapports |

# 3. Description détaillée

*Note: Pour cause de problème de gestion de caractères avec Visual Paradigm, la majeure partie des mots contenus dans les diagrammes ont été écrits en anglais.*

## *3.1 Spécification des cas d'utilisation*

### *3.1.1 Le modèle "brut" des cas d'utilisation*

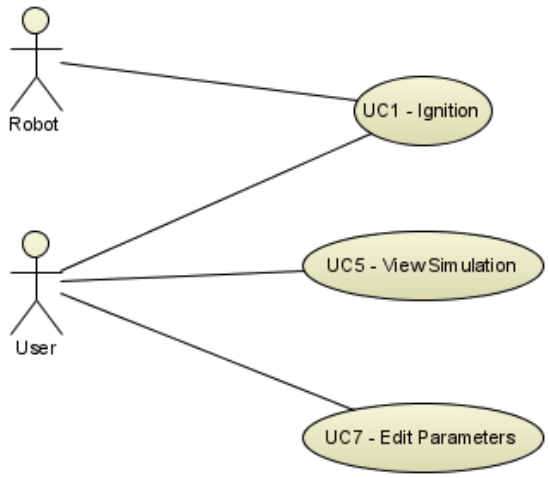


*Illustration 1: Modèle "brut" des cas d'utilisation*

### 3.1.2 Le modèle "détaillé" des cas d'utilisation

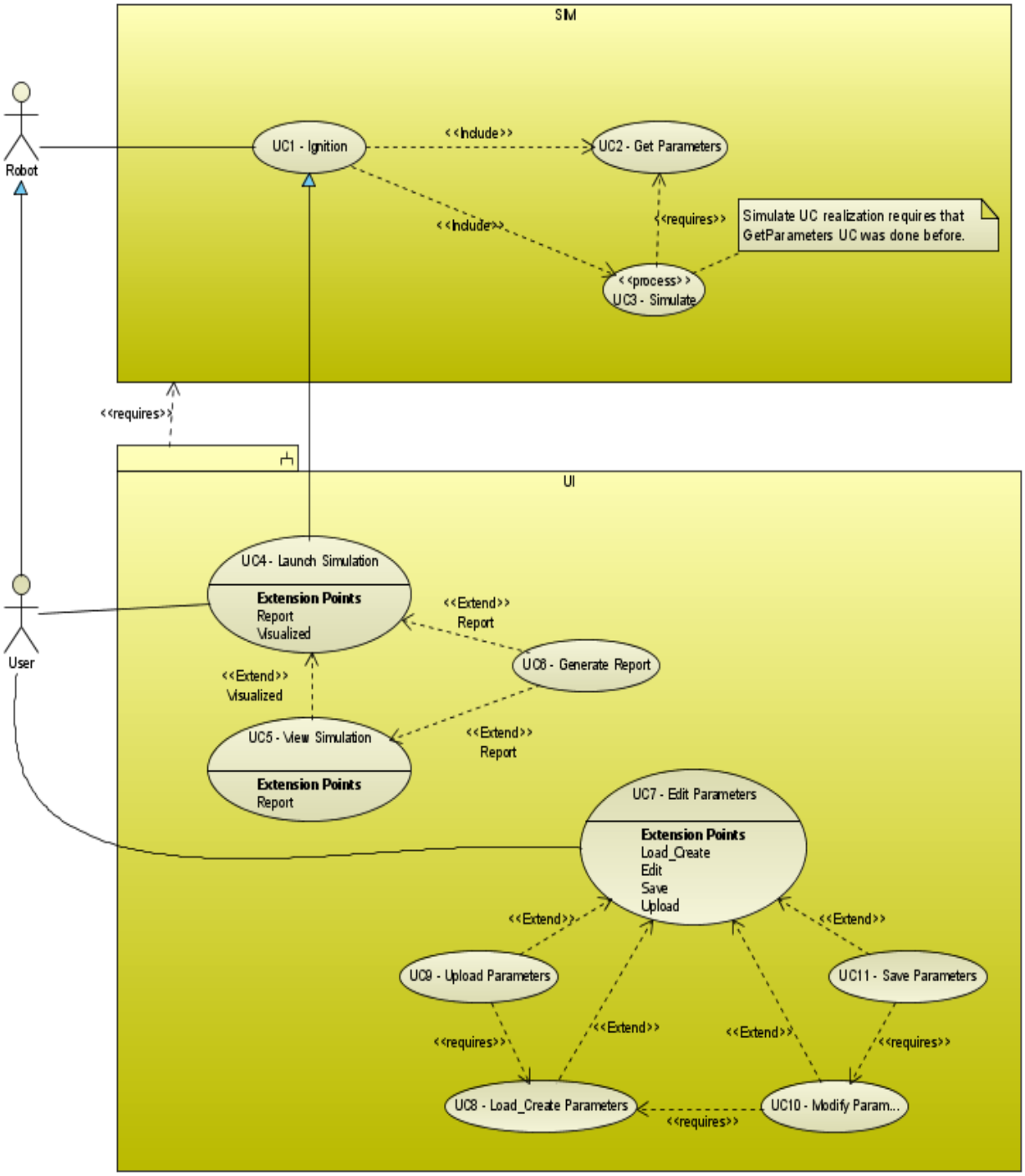


*Illustration 2: Modèle "détaillé" des cas d'utilisation*

## UC1- Ignition

**Module :** SIM (simulateur)

**Mode :** Interactif (user goal)

**Acteurs :** Robot (primaire), Utilisateur (généralisation de l'acteur Robot)

**Résumé :** Ce cas d'utilisation décrit  la mise en marche du simulateur.

**Pré-condition(s) :** le simulateur ne doit pas être déjà activé.

**Post-condition(s) :** Le simulateur génère des données (le robot se déplace).

**Scénario nominal**

1. Le robot (ou l'utilisateur) active le programme du simulateur.
2. Le simulateur réalise le cas d'utilisation [UC2 – Get Parameters].
3. Le simulateur réalise le cas d'utilisation [UC3 – Simulate].
4. Expiration de la simulation après un certain délai.

**Extension(s)**

**a : À tout moment, le système tombe en panne*.

Le robot (ou l'utilisateur) redémarre le système.

*4a : Arrêt manuel*

1. Le robot (ou l'utilisateur) arrête le système.

**Besoins spéciaux**

- Interrupteur physique et interrupteur logique.
- Témoin d'activation et de bon fonctionnement du simulateur.

**Cas ouvert(s)**

- Doit-on envisager une sauvegarde continu de l'état de la simulation pour que, en cas de panne, elle puisse être relancé plutôt que réinitialisé ? Puissance computationnelle nécessaire à évaluer.

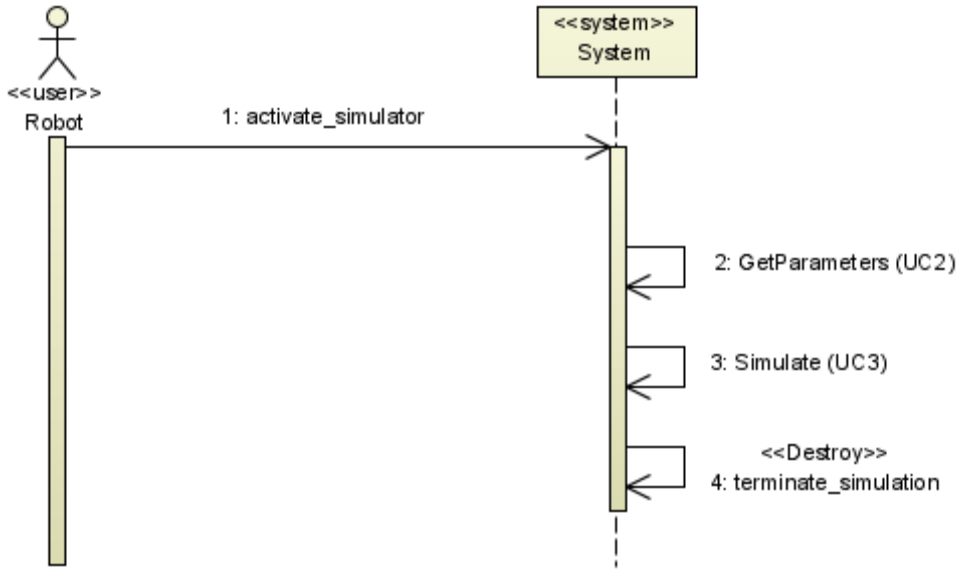


*Illustration 3: UC1 – Ignition - Main-Scenario*

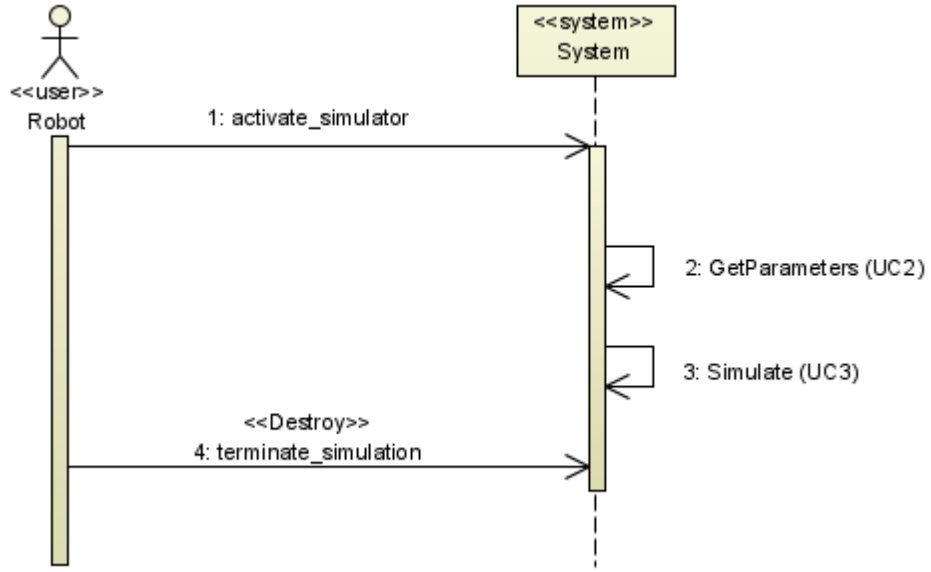


*Illustration 4: UC1 – Ignition - Extension - 4a*

## UC2 – Get Parameters

**Module :** SIM (simulateur)

**Mode :** Autonome (interactif avec le fichier de données)

**Résumé :** Ce cas d'utilisation décrit  la récupération des paramètres.

**Acteurs :** Simulateur (~primaire, car autonome)

**Pré-condition(s) :** Le programme doit avoir été activé.

**Post-condition(s) :** Activation de la simulation (UC3).

**Scénario nominal**

1. Le simulateur charge les données.
3. Le simulateur crée un objet contenant les paramètres.
4. Le simulateur crée un objet contenant les événements du *monde*.
5. Le simulateur signale le bon déroulement du processus (témoin lumineux fixe).

**Extension(s)**

*1a : Fichier introuvable ou erroné*

1. Émission d'une erreur (témoin lumineux clignotant).
2. Arrêt du simulateur.

**Cas ouvert(s)**

- Quel type d'encodage de données pour la désérialisation (poids du fichier et rapidité du traitement): XML, binaire, hybride, autre ?

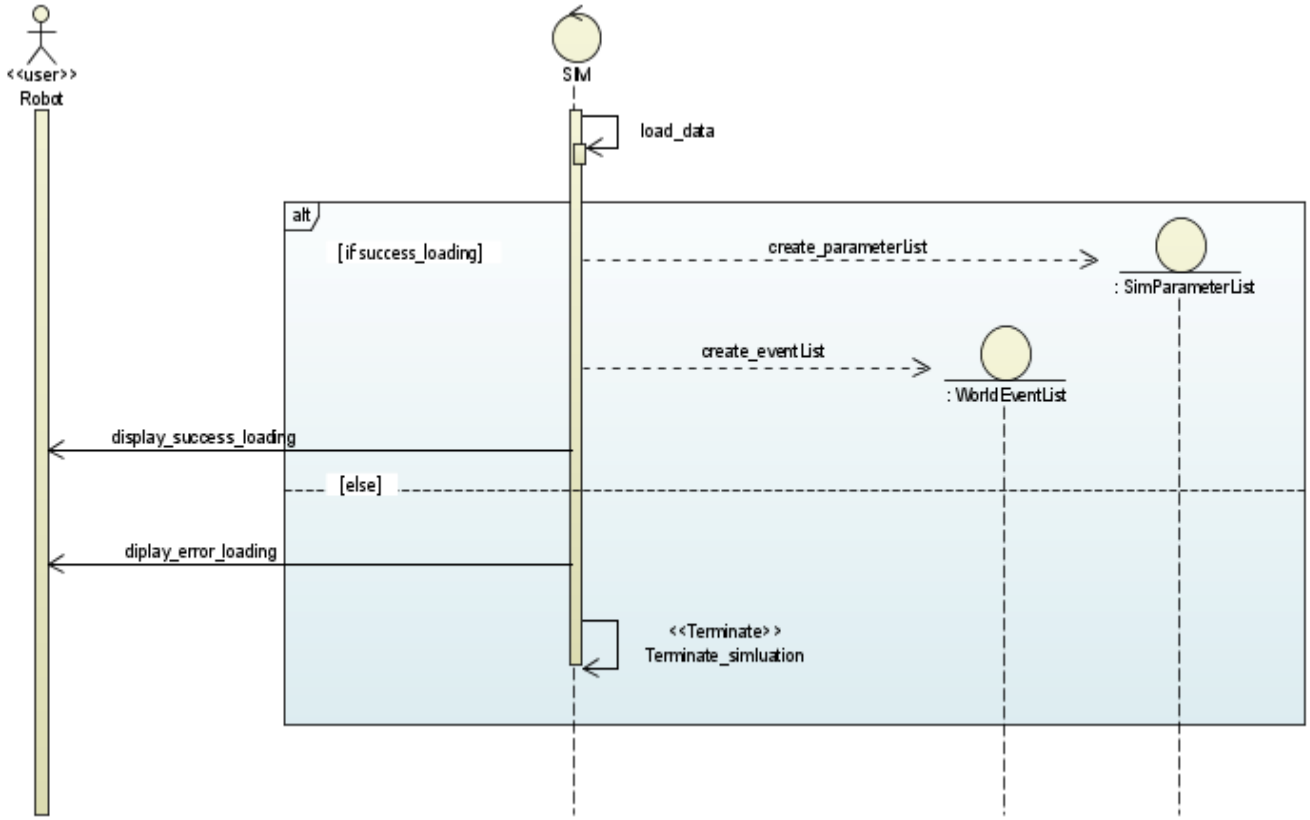


*Illustration 5: UC2 – Get Parameters - Main-Scenario & Extension - 1a*

## UC3 - Simulate

**Module :** SIM (simulateur)

**Mode :** autonome

**Résumé :** Ce cas d'utilisation décrit  la simulation en tant que tel.

**Acteurs :** Simulateur (~primaire, car autonome)

**Pré-condition(s) :** Les paramètres doivent être chargés (UC2).

**Post-condition(s) :** la simulation se poursuit jusqu'à son terme (arrêt automatique ou manuel).

**Scénario nominal**

1. Le simulateur crée un monde
2. Le simulateur crée un agent atomique (agent minimal)

*>> Répéter jusqu'à ce que le monde soit rempli*

3. Les agents atomiques sont agrégés pour former des agents composites.

*>> Répéter jusqu'à ce que le ou les agents les plus englobants soient trouvés*

4. Le simulateur insère un événement dans le monde

*>> Répéter les étapes 3 et 4 jusqu'à ce qu'à atteindre la stabilité du système (i.e. boucle infinie pour les systèmes auto-oscillants car la stabilité n'est jamais atteinte pour de tel système; c'est un équilibre dynamique)*

**Extension(s)**

*aucune*

**Cas ouvert(s)**

- Tout est à penser pour l'algorithme. Ce cas d'utilisation est donc une proposition temporaire.

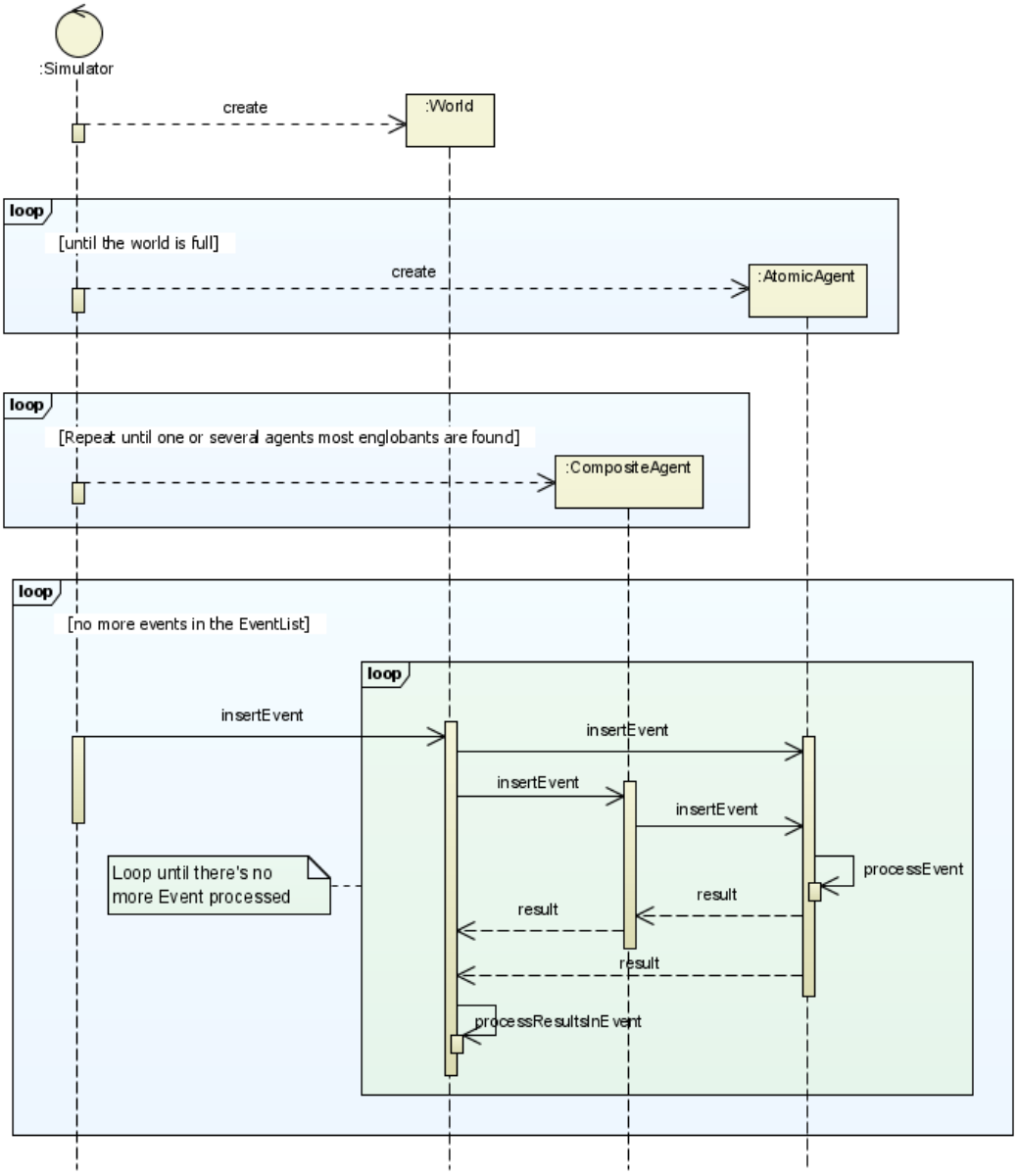


*Illustration 6: UC3 – Simulate - Main-Scenario*

## UC4 – Launch Simulation

**Module :** UI (interface utilisateur)

**Mode :** Interactif (user goal)

**Acteurs :** Utilisateur

**Résumé :** Ce cas d'utilisation décrit l'activation du simulateur via l'interface utilisateur.

**Pré-condition(s) :** Le simulateur ne doit pas être déjà activé.

**Post-condition(s) :** Visualiser la simulation ou le rapport.

**Scénario nominal**

1. L'utilisateur active l'interface utilisateur (UI).
2. Le programme de l'UI se connecte au simulateur.
3. L'utilisateur choisi un mode de visualisation (UC5, UC6).
4. L'utilisateur lance le simulateur depuis l'UI.
5. UI lance le simulateur.
6. UI lance une fenêtre de visualisation (correspondant au choix UC5 ou UC6) pour afficher les résultats (la simulation)

**Extension(s)**

**a : À tout moment, le système tombe en panne*.

L'utilisateur redémarre le système.

**b : arrêt manuel*

1. L'utilisateur arrête la simulation.

2a. Quitter le programme

2b. Reprendre le scénario nominal en (3).

*2a : échec de connexion au simulateur*

1. Le programme signale l'échec de connexion à l'utilisateur.

*3a : UC6 en extension du UC5*

1. Si l'utilisateur choisit le UC5, il peut aussi y associer le UC6

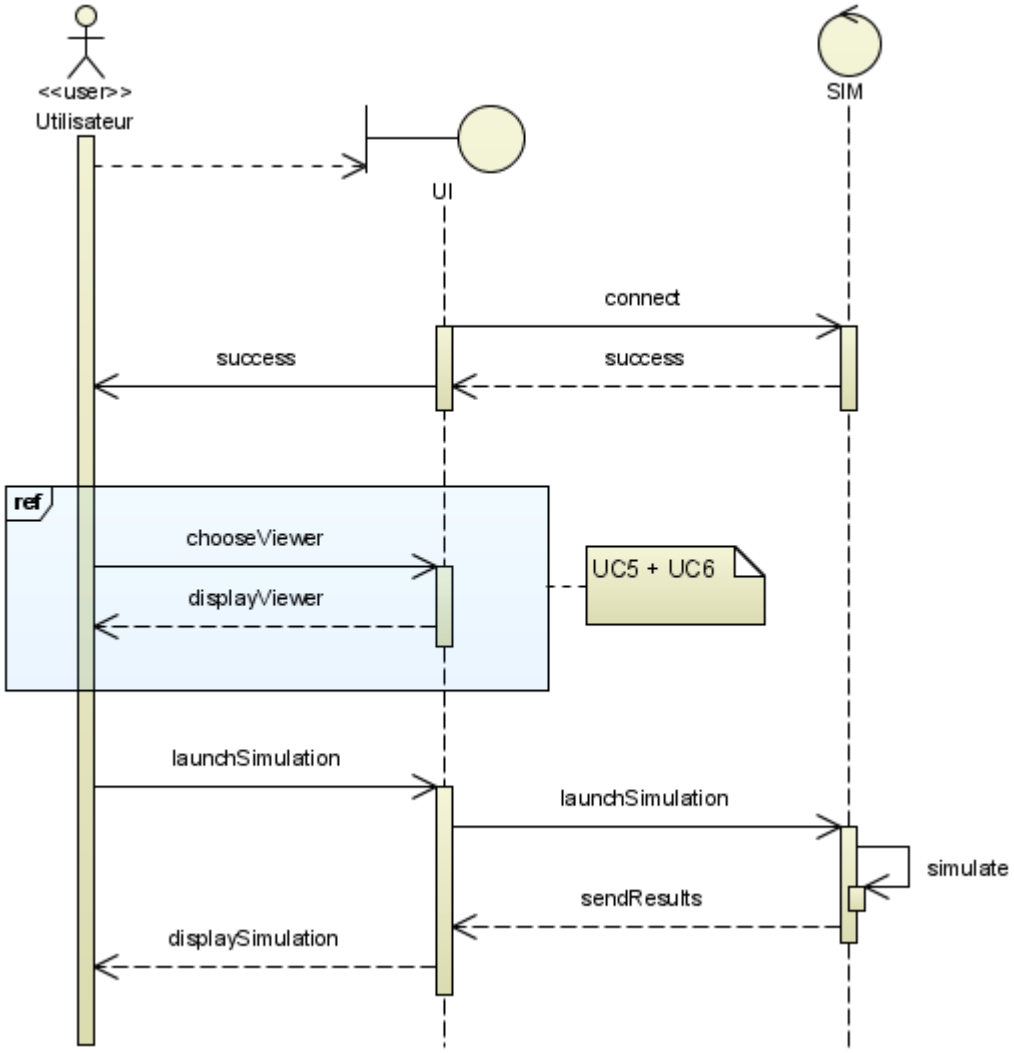


*Illustration 7: UC4 – Launch Simulation - Main-Scenario*

## UC5 – View Simulation

**Module :** UI (interface utilisateur)

**Mode :** Interactif (user goal)

**Acteurs :** Utilisateur

**Résumé :** Ce cas d'utilisation décrit  l'activation d'une interface de simulation.

**Pré-condition(s) :** l'UI doit être connectée au simulateur.

**Post-condition(s) :** voir la fenêtre de visualisation.

**Scénario nominal**

1. L'utilisateur choisi un type de visualisation : 2D, 3D, paramètres d'affichage dont « générer un rapport » (UC6).
2. Le programme construit l'interface
3. Le programme affiche l'interface

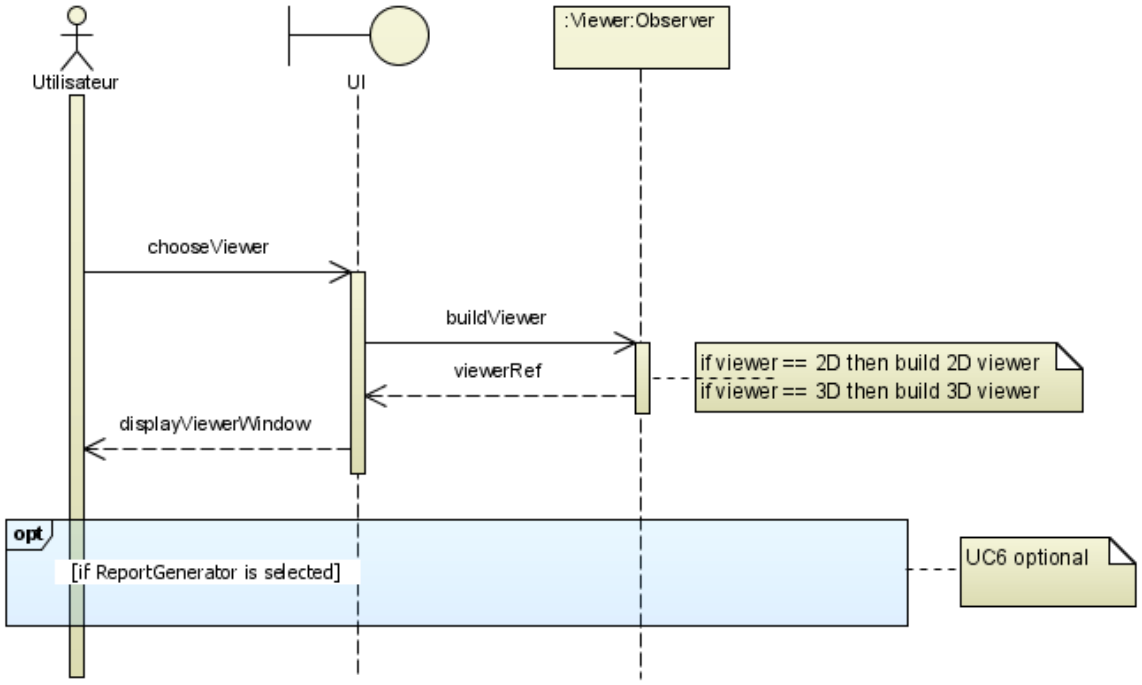


*Illustration 8: UC5 – View Simulation - Main-Scenario*

## UC6 – Generate Report

**Module :** UI (interface utilisateur)

**Mode :** Interactif (user goal)

**Acteurs :** Utilisateur

**Résumé :** Ce cas d'utilisation décrit  l'activation du générateur de rapport d'une simulation.

**Pré-condition(s) :** l'UI doit être connectée au simulateur.

**Post-condition(s) :** voir le rapport.

**Scénario nominal**

1. L'utilisateur choisi de « générer un rapport »
   (directement, ou comme option du UC5).
2. Le programme génère le rapport.
3. Le programme affiche le rapport.

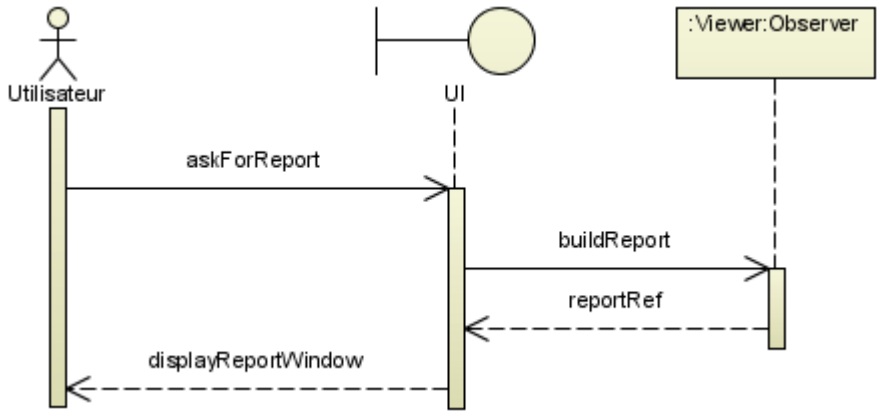


*Illustration 9: UC6 – Generate Report - Main-Scenario*

## UC7 – Edit Parameters

**Module :** UI (interface utilisateur)

**Mode :** Interactif (user goal)

**Acteurs :** Utilisateur

**Résumé :** Ce cas d'utilisation décrit l'activation du menu d'édition.

**Pré-condition(s) :** aucune.

**Post-condition(s) :** aucune.

**Scénario nominal**

1. L'utilisateur active le menu d'édition des paramètres.
2. Le programme affiche le menu.
3. L'utilisateur choisit un item du menu.
4. Le programme exécute la commande correspondante à l'item activé.
5. Le programme affiche le résultat correspondant à la commande.

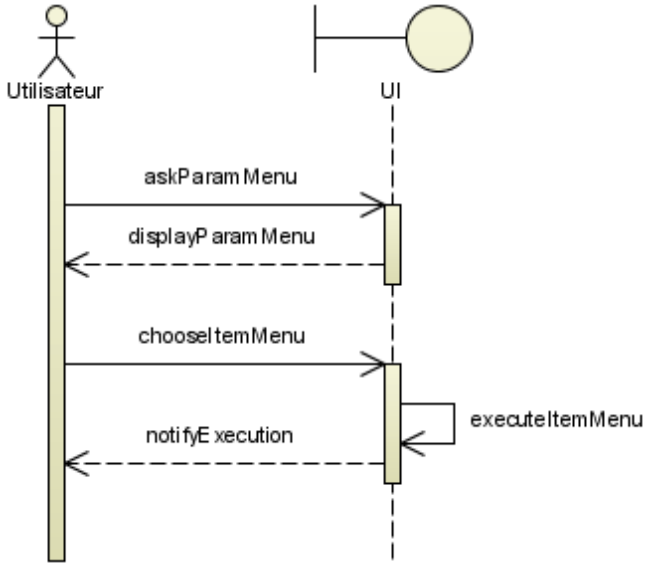


*Illustration 10: UC7 – Edit Parameters - Main-Scenario*

## UC8 – Load-Create Parameters

**Module :** UI (interface utilisateur)

**Mode :** Interactif (user goal)

**Acteurs :** Utilisateur

**Résumé :** Ce cas d'utilisation décrit le chargement d'un fichier de paramètres.

**Pré-condition(s) :** menu d'édition des paramètres activé.

**Post-condition(s) :** fichier de paramètres affiché.

**Scénario nominal**

1. L'utilisateur demande au programme de charger un fichier de paramètres.
2. Le programme charge le fichier
3. Le programme affiche le fichier

**Extension(s)**

*1a : Création d'un fichier de paramètres*

1. Au lieu de charger un fichier de paramètres, l'utilisateur demande d'en créer un.
2. Le programme génère un fichier de paramètres.

*2a : Erreur de chargement ou fichier inexistant*

1. Le programme retourne une notification d'erreur.

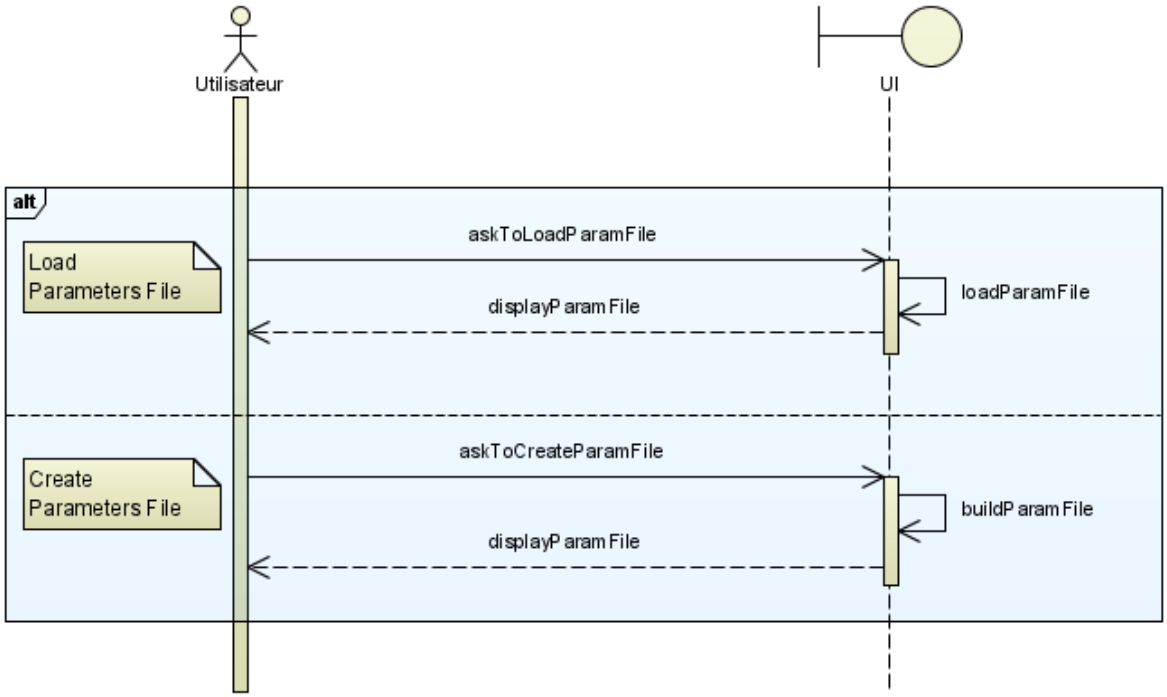


*Illustration 11: UC8 –Laod-Create Parameters - Main-Scenario*

## UC9 – Upload Parameters

**Module :** UI (interface utilisateur)

**Mode :** Interactif (user goal)

**Acteurs :** Utilisateur

**Résumé :** Ce cas d'utilisation décrit la sauvegarde d'un fichier de paramètres au niveau de l'emplacement du simulateur.

**Pré-condition(s) :** fichier de paramètres affiché (UC8).

**Post-condition(s) :** aucune.

**Scénario nominal**

1. L'utilisateur demande au programme de sauvegarder le fichier de paramètres dans la le répertoire du simulateur.
2. Le programme enregistre le fichier dans l'emplacement approprié.

**Extension(s)**

*2a : Emplacement inexistant*

1. Le programme retourne une notification d'erreur.

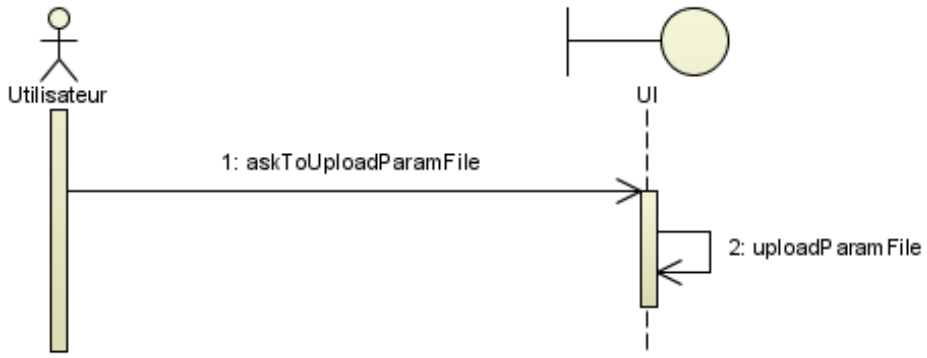


*Illustration 12: UC2 – Upload Parameters - Main-Scenario*

## UC10 – Modify Parameters

**Module :** UI (interface utilisateur)

**Mode :** Interactif (user goal)

**Acteurs :** Utilisateur

**Résumé :** Ce cas d'utilisation décrit la modification d'un fichier de paramètres.

**Pré-condition(s) :** fichier de paramètres affiché (UC8).

**Post-condition(s) :** aucune.

**Scénario nominal**

1. L'utilisateur modifie des paramètres.
2. Le programme actualise les modifications.
3. L'utilisateur sauvegarde le fichier de paramètres (UC9 ou UC11)

**Extension(s)**

*3a : Fermeture de la fenêtre de paramétrage avec des modifications non sauvegardées*

1. Le programme propose de sauvegarder le fichier avant la fermeture de ce dernier.

2a. L'utilisateur quitte sans sauvegarder

2b. (UC11)

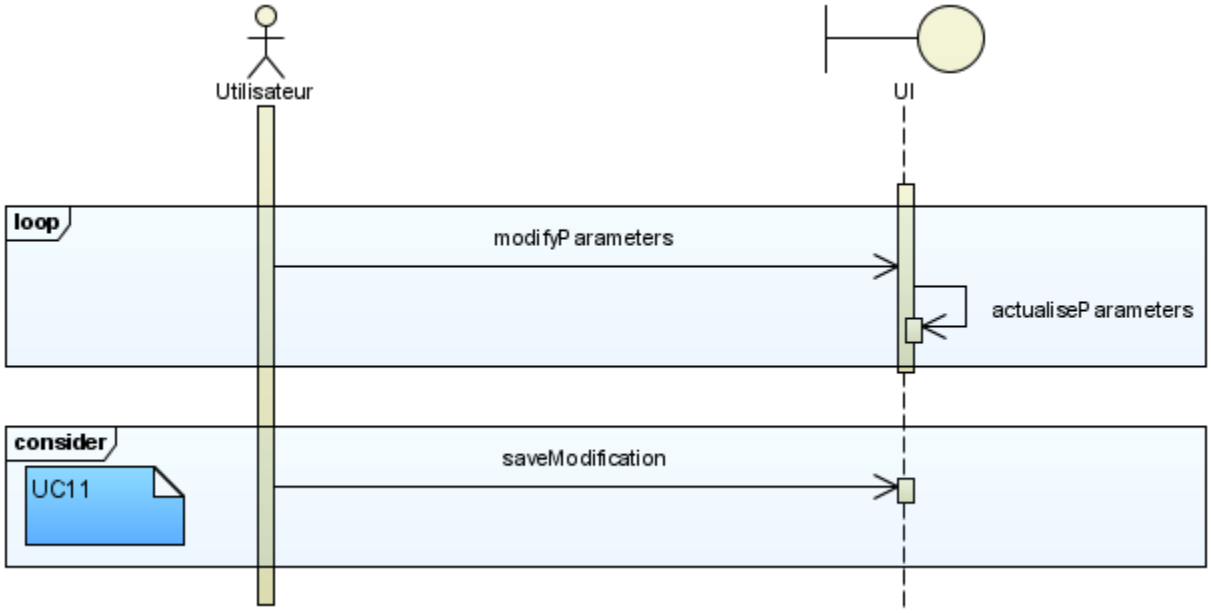


*Illustration 13: UC2 – Modify Parameters - Main-Scenario*

## UC11 – Save Parameters

**Module :** UI (interface utilisateur)

**Mode :** Interactif (user goal)

**Acteurs :** Utilisateur

**Résumé :** Ce cas d'utilisation décrit la sauvegarde d'un fichier de paramètres à un emplacement spécifié par l'utilisateur.

**Pré-condition(s) :** fichier de paramètres affiché (UC8); modifications apportées au fichier.

**Post-condition(s) :** aucune.

**Scénario nominal**

1. L'utilisateur demande au programme de sauvegarder le fichier de paramètres.
2. Le programme demande où enregistrer le fichier.
3. L'utilisateur indique l'emplacement de sauvegarde.
2. Le programme enregistre le fichier dans l'emplacement approprié.

**Extension(s)**

*2a : Chemin d'emplacement erroné*

1. Le programme retourne une notification d'erreur.

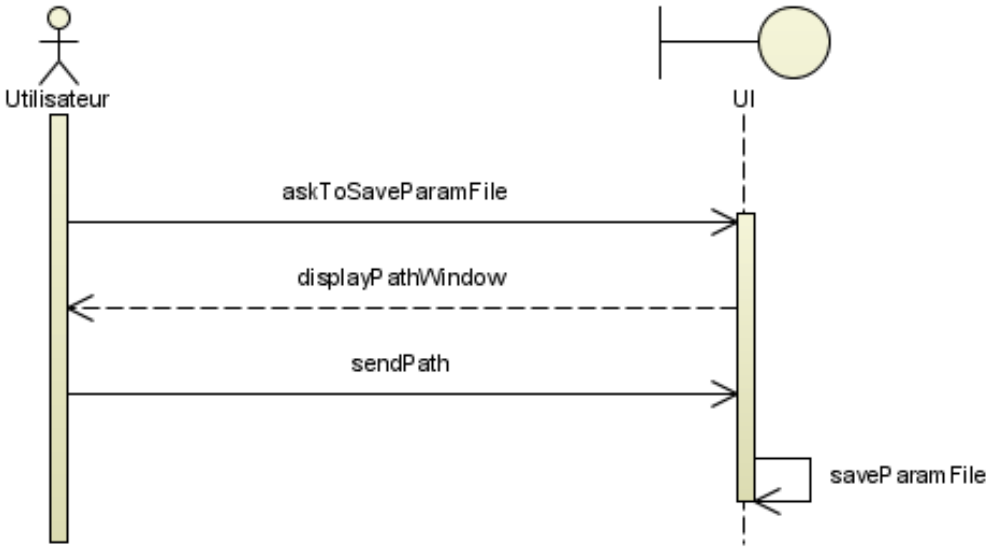


*Illustration 14: UC11 – Save Parameters - Main-Scenario*

## *3.2 Spécifications conceptuelles*

### 3.2.1Modèle objet

Tous les modèles présentés ici concerne le simulateur en tant que tel. Considérant que le coeur du projet n'est pas encore défini (*), les modèles dynamiques (d'interaction et d'organisation) ne sont donné qu'à titre indicatif afin d'alimenter la réflexion et en ce sens ne suivront pas nécessairement les modèles objets qui ne font que présenter l'idée générale du simulateur (d'où le fait que, même dans le modèle objet « détaillé », les attributs de classes ne sont pas nommés).

(*) Le choix d'un modèle de développement, des langages de programmation, des standards et des spécifications n'est pas encore arrêté.

#### 3.2.1.1 Modèle objet "brut"

Ce modèle décrit uniquement le système (le module SIM, i.e. le simulateur).

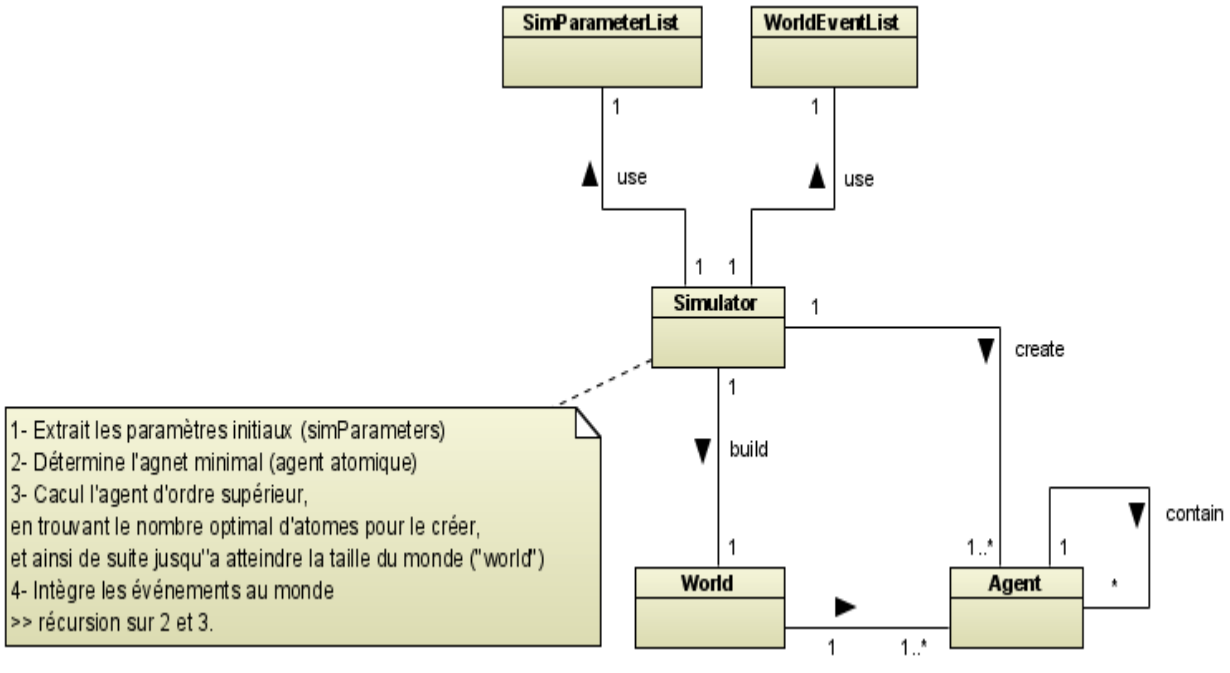


*Illustration 15: Modèle objet "brut"*

Le simulateur (« *Simulator* ») utilise la liste de paramètres pour construire l'environnement et les agents atomiques. Il n'a pas encore été décidé si les agents

atomiques s'agrégeraient par eux-mêmes pour former des agents composites ou si ce serait le simulateur qui le ferait.

Une fois l'étape de création terminée, le simulateur injecte des événements, extraits de la liste correspondante, dans le monde.

#### 3.2.1.2 Modèle objet "détaillé"

Ce modèle décrit à la fois le module SIM (le simulateur) et le module satellite UI (l'interface utilisateur).

Le simulateur en tant que tel est contenu dans l'espace de nommage (*package* dans la suite du document) SIM. L'interface utilisateur est totalement séparée du simulateur afin de rendre ce dernier portable dans différents environnements( robot entre autre). En fait ces deux package sont de applications distinctes.

Dans le package UI, on trouve deux autres espaces; celui de la gestion de la visualisation des simulations et celui de l'édition des paramètres.

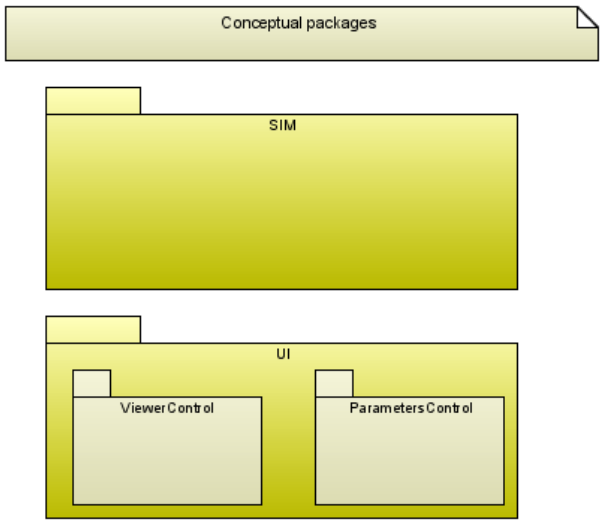


*Illustration 16: Architecture logique*

Le modèle objet « détaillé » présente le détail de ces *packages*. Il met en évidence les différents « *patterns* » utilisés pour gérer l'interaction sur les éléments communs aux deux applicatifs.

Dans SIM, le « *Deserializer* » permet de traiter le fichier de paramètres. Cette fonction est aussi supportée dans UI, quoique d'un manière et dans un objectif différents, par le package « *ParametersControl* ».

Le pattern « *Observer* », quant à lui, permet de mettre en communication le « *Simulator* » de SIM avec le « *ViewerControl* » de UI.

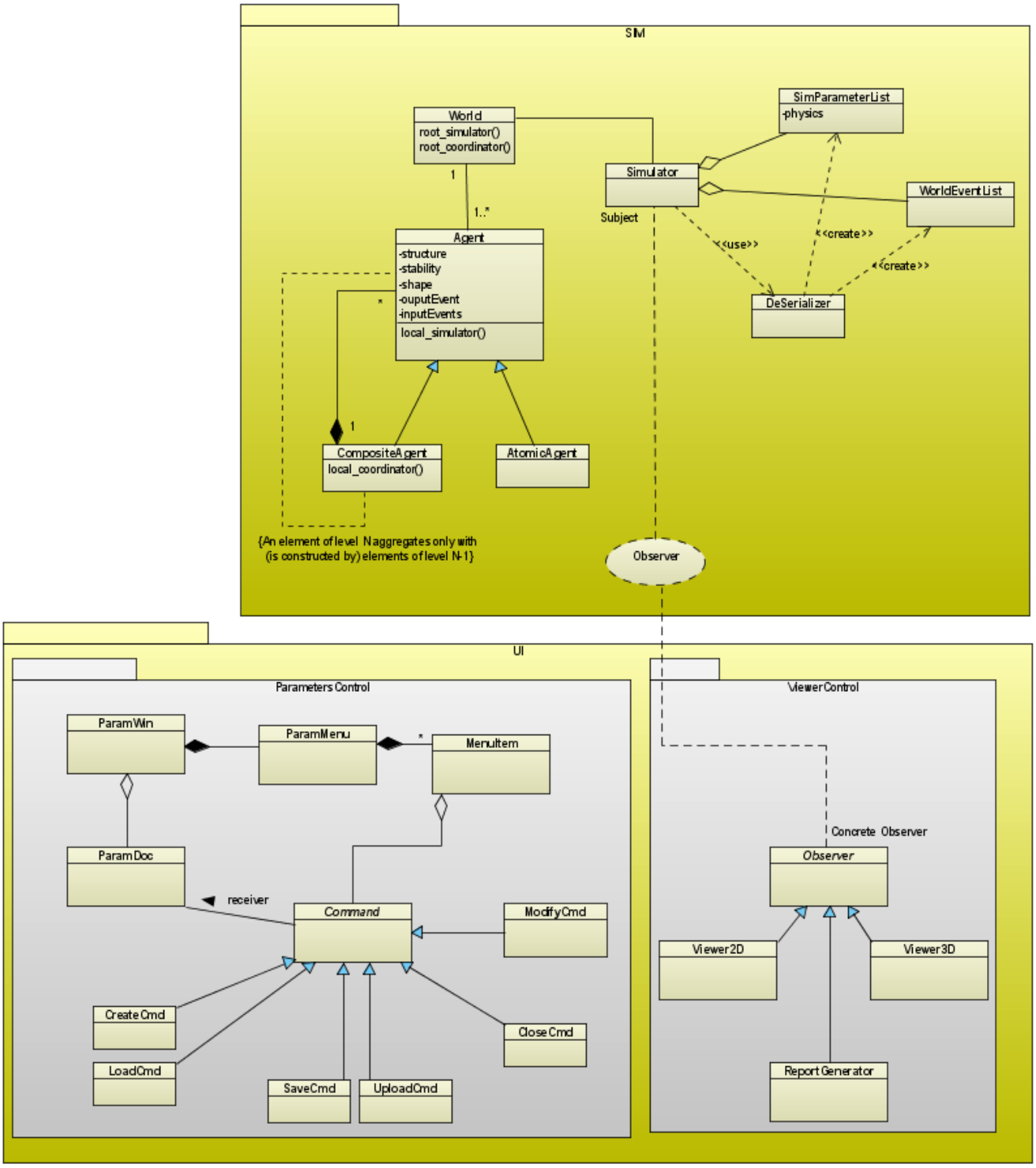


*Illustration 17: Modèle objet "détaillé"*

### *3.2.2 Agents et relation de couplage*

Cette section vise à introduire le concept d'agent dans le cadre de ce projet ainsi que le mécanisme d'agrégation structurale des agents.

- **Agent élémentaire**

L'agent élémentaire est l'agent de base à partir duquel sont dérivés les agents atomiques et composites. c'est un agent dit réactif pulsionnel, i.e. Un agent pulsionnel aura une mission fixée (par exemple, s'assurer qu'un réservoir reste toujours suffisamment rempli) et déclenchera un comportement s'il perçoit que l'environnement ne répond plus au but qui lui était affecté (le niveau du réservoir est trop bas).

Un agent admet en entrée un flux d'événements (sommés) et émet en retour un flux d'événements (singuliers). Le flux d'entrée génère, à partir d'un certain seuil, une instabilité de l'agent qui tentera de résoudre alors sa propre problématique. Le flux de sortie correspond aux répercutions de l'activité de l'agent sur son environnement proximal. Ici *proximal* ne signifie pas *local*. Il est entendu par ce terme que l'activité de l'agent peut diffuser, i.e. perturber des agents directement connexe mais aussi plus distant (remontée et descente de quelques branches dans un arbre hiérarchique)

L'agent cesse d'émettre un flux lorsqu'il rétabli son équilibre, i.e. sa structure, laissant ainsi apparaître sa forme optimale.


*Illustration 18: Agent élémentaire*

- **Agent atomique**

Il est en tout point identique l'agent élémentaire car il est toujours un noeud terminal.

Le comportement d'un agent peut être décrit sommairement à partir de celui d'un agent atomique. Un tel agent est dans un état dit *passif*, lorsqu'il n'est pas perturbé par un événement qui lui est externe (*inputEvent*) et qu'il est dans un état stable (que sa

problématique existentielle est résolue, relativement à l'environnement dans lequel il se situe). Lorsque l'environnement change, l'agent passe dans un état actif qui correspond à une recherche de stabilité. Cette dernière et source de changement interne qui se répercutent sur l'environnement avoisinant (*ouputEvent*).

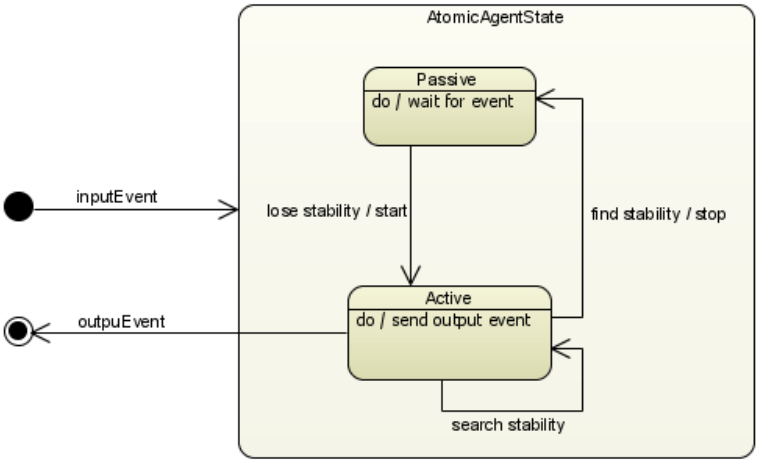


*Illustration 19: Diagramme d'états d'un agent atomique*

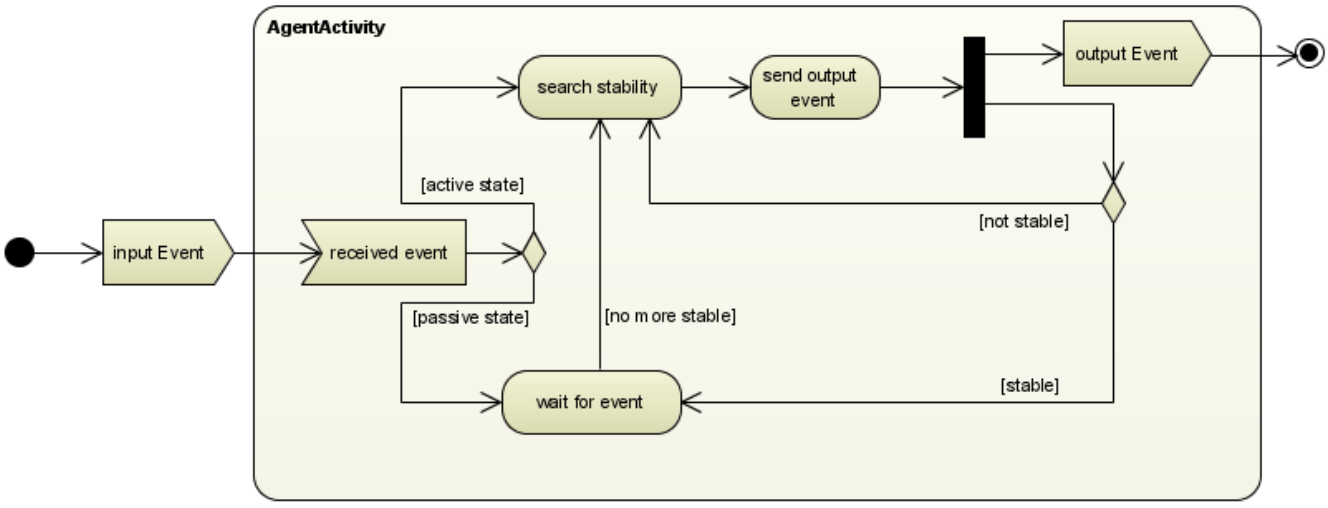


*Illustration 20: Diagramme d'activités d'un agent*

- **Agent composite**

Bien que similaire à l'agent atomique, il est suppléé d'un *module de coordination* qui a pour double fonction de (1) lui permettre de communiquer avec ses agents agrégés mais aussi (2) de faciliter (centraliser) la communication entre ces derniers. En d'autres termes, le module de coordination facilite la transmission des communications à l'intérieur d'un agent composite, mais aussi de celle reçu de son environnement ou émise vers ce dernier.

L'*illustration 19* présente un agent composite lui même composé d'un autre agent composite et d'un agent atomique. Bien-sûr, un agent composite peut-être réalisé par plus de deux agents mais, pour la clarté de la description, nous nous limiterons au cas de deux agents en réalisant un troisième.

Ce premier schéma montre la récurrence des événements de sortie sur les entrées des agents agrégés. Ces agents sont donc fortement « liés », ce qui entraîne une complexification de la recherche de stabilité au niveau de l'agent composite résultant de cette agrégation. Bien que la fonction des agents reste la même (recherche de stabilité, optimisation des imperfections), il en résulte un saut qualitatif entre le modèle de recherche de stabilité à l'échelle de premier ordre (les agents agrégés) et celui de l'échelle de second ordre. C'est ce que nous nommons la *covariance d'échelle*. où le principe de « recherche de stabilité » perdure, mais les mécanismes, ou modèles, varient.

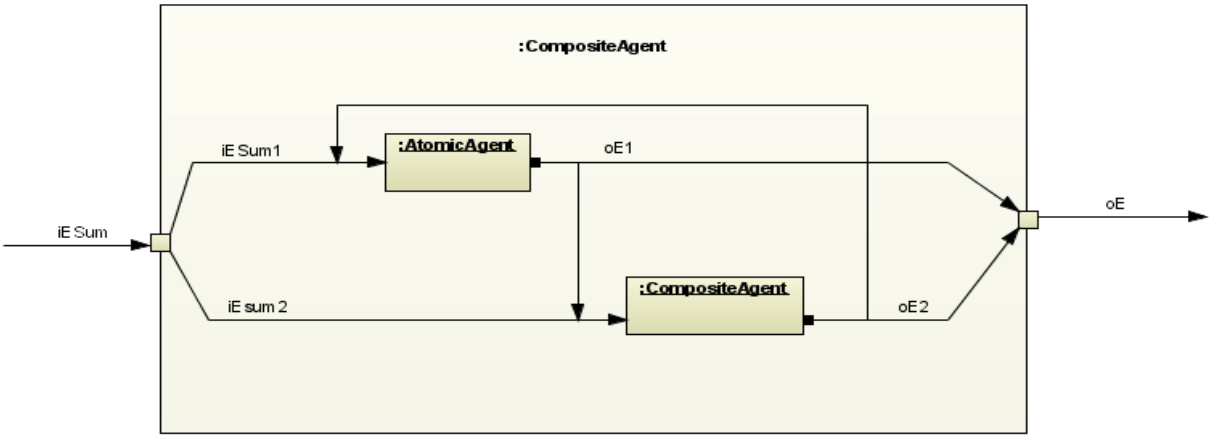


*Illustration 21: Exemple d'agent composite*

Chaque agent composite possède un *module de coordination* qui lui permet de gérer la communication entre lui et les agents qui le composent, mais aussi entre ces derniers.

Comme dans le cas de l'agent atomique, le comportement de l'agent composite est réactif et ne vise que la recherche d'une stabilité structurelle (représenté par l'émergence d'une forme stable). À la différence du premier, l'agent composite recherche sa propre stabilité par la sommation de leur événements de sortie avec l'événement d'entrée extérieur (provenant de l'environnement) desquels il recherche une adéquation, i. e. une stabilisation `la fois à son niveau (second ordre) mais aussi au niveau des agents agrégés (premier ordre) par émission de ses événements de sortie.

Dès lors, il apparaît une récursivité imbriquée, ou plus simplement un bouclage d'ordre en ordre et ce aussi bien verticalement (niveau d'ordre) horizontalement (agent du même ordre, du même niveau). Aussi, les agents d'un même niveau peuvent être de différent type, comme dans notre exemple.

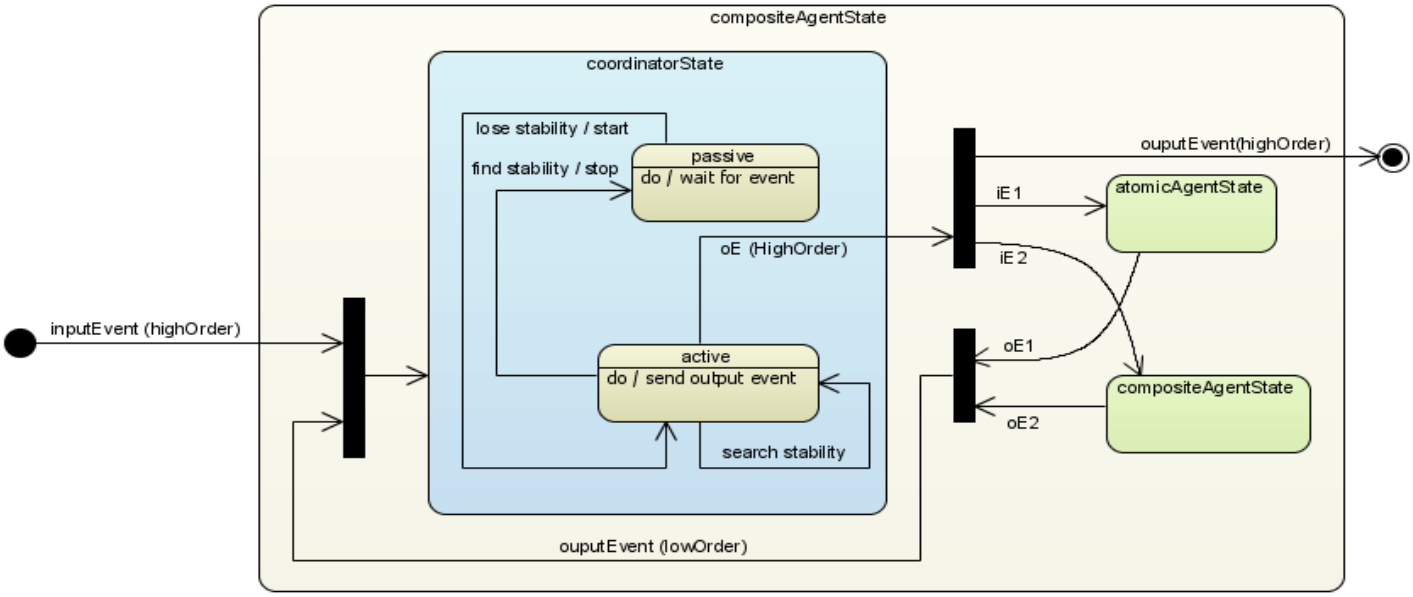


*Illustration 22: Diagramme d'états d'un agent composite*

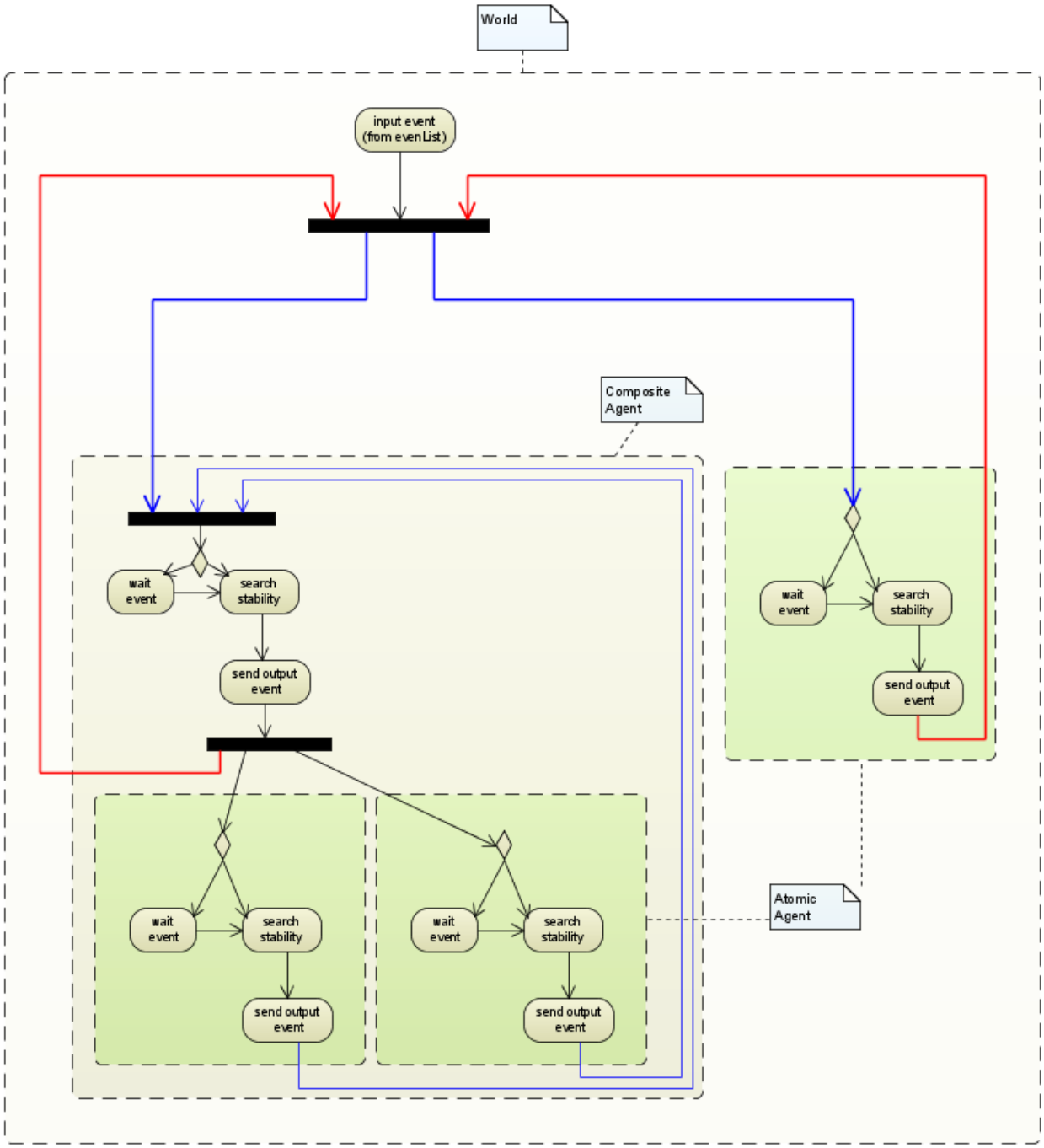


*Illustration 23: Diagramme d'activités du monde*

## *3.3 Exigences d'opérations, de communications et de performance*

Puisque l'application, tel qu'envisagée actuellement, est monoposte, il n'y a pas d'exigence pour ce qui est du nombre de terminaux à supporter ou du nombre d'utilisateurs simultanés.

Pour ce qui est de la quantité d'information et du nombre de transactions à traiter par unité de temps, il est raisonnable d'envisager une possible évolution, et donc une ouverture du système, vers le calcul parallèle et distribué.

Aussi, pour ce qui est de la sauvegarde des informations et des visuels des simulations, il faudra envisager des moyens de stockage efficients.

## *3.4 Exigences logiques de bases de données*

Actuellement, il n'est pas envisagé d'utiliser de bases de données, mais plutôt des fichier compressé de type AVI ou DivX pour les animations. Rien n'a été décidé en ce qui concerne les données brutes qui seront peut-être des données non persistantes.

## *3.5 Contraintes de conception*

Les contraintes de conceptions devront garantir l'ouverture de l'application à un développement pour le calcul parallèle et distribué.

Pour ce qui est du format des rapports, il est envisager de les produire suivant la norme XML afin d'en facilité la portabilité (et accessoirement la présentation).

## *3.6 Exigences non-fonctionnelles*

| ***Contraintes*** | ***Descriptif*** |
|---|---|
| Temps de réponse | L'interface de visualisation doit pouvoir rafraîchir rapidement les données l'écran (la simulation). |
| Disponibilité | L'application et le matériel qui la supporte doit être accessible 7 jours sur 7 et 24 heures sur 24, par exemple en cas de simulation de longue durée ou en cas d'entretien de la machine (relance de simulation) |
| Intégrité | L'application et le matériel qui la supporte doivent être protégé de tout vandalisme, par exemple les installant dans un local à accès |

| *Contraintes* | *Descriptif* |
|---|---|
| | fortement restreint. |
| Concurrence | non-applicable (mono-utilisateur) |

## *3.7 Dictionnaire des données (Glossaire)*

Le dictionnaire de données n'inclue pas dans le présent document les formats de données, règles de validation et alias. Nous ferons, pour cette première itération, une simple liste descriptive des éléments de modélisation.

**Robot.** utilisateur mécanique ou virtuel (miniBot) exploitant le simulateur pour, par exemple, se mouvoir dans un environnement (réel ou virtuel).

**Utilisateur.** Chercheur au sens large (philosophe, informaticien, ingénieur, autre) utilisant le simulateur à des fins expérimentales.

**SIM.** L'application dénommé *Ka* et dont l'objectif est la simulation de système dynamique non-linéaire oscillant ou auto-oscillant (e.g. simulation de systèmes chimiques)

**Simulator.** C'est le coeur du simulateur. Il contient la partie algorithmique permettant de générer la simulation.

**World.** C'est l'environnement paramétré dans lequel se déploient les agents. c'est dans ce « monde » qu'on lieu les événements (provenant soit des agents soit de la liste pré-établie d'événements)

**Agent.** Un agent est une entité partiellement autonome, située (vis-à-vis de l'environnement « *world* ») et en interaction avec ce dernier (et donc avec d'autres agents). Un agent, dans le cadre de cette application, est de type réactif. Il est à la fois (1) tropique, dans le sens où il ne réagit que par rapport à l'état de son environnement local et (2) pulsionnel car il déclenchera un comportement *d'optimisation de ses imperfections* s'il perçoit que l'environnement n'est plus adéquat avec son état. Dit autrement, l'agent dont l'état courant ne ne résout plus sa problématique interne 9par un changement de l'environnement, tentera de retrouver son équilibre, i.e. de résoudre sa problématique interne. c'est un comportement hautement dynamique. Aussi, il existe deux type d'agents, les agents atomiques et les agents composites. Un

agent atomique n'a pas d'autre composant que lui-même alors qu'un agent composite est un agrégat d'agents composites ou atomiques .

**WorldEventList.** C'est la liste des événement injecté dans le monde. Un événement est l'équivalent d'une perturbation au sein du système. Un événement ne permet pas nécessairement d'initialiser le comportement global du système (e.g. les systèmes auto-oscillants) mais de le faire évoluer via des perturbation. Le système (« *world* ») évoluant est un système entrain de résoudre sa problématique (lié à l'incorporation d'une perturbation) et en se sens est un système avec des propriétés computationnelle spatiales.

**SimParameterList.** C'est la liste des paramètres permettant au simulateur (« S*imulator* ») de générer l'environnement et les agents et d'actualiser le premier (les agents, puisque réactifs, s'actualisent d'eux-même).

**UI.** Application de type interface utilisateur permettant de visualiser les simulations, de les paramétrer et de générer des rapports.

**ParametersControl.** Ensemble des commandes permettant d'éditer les paramètres (et donc de créer différent jeu de paramètres).

**ViewerControl.** Il transforme les données d'une simulation en un visuel (2D ou 3D).

# 4. Informations complémentaires

## *4.1 Index*

## *4.2 Annexes*

*Présentation*

Jc_dic9250_Presentation.pdf

*R. Duboz, E. Ramat*

XML pour la représentation des données, du couplage de modèles et des expériences de simulations

http://www-lil.univ-littoral.fr/%7Eduboz/Papers/RI-LIL-2003-01.ps

*Bernard P. Zeigler*

DEVS Today: Recent Advances in Discrete Event-Based Information Technology

http://www.acims.arizona.edu/EDUCATION/ECE575Fall03/Notes/DevsToday2Col.pdf

*Rajanikanth Jammalamadaka*

Activity Characterization of Spatial Models: Application to Discrete Event Solution of Partial Differential Equations

http://www.acims.arizona.edu/PUBLICATIONS/PDF/MSThesisRaj.pdf

*Nutaro, J., Hammonds, P.*

Combining the Model/View/Control Design Pattern with the DEVS Formalism to Achieve Rigor and Reusability in Distributed Simulation

http://www.scs.org/pubs/jdms/vol1number1/article02.pdf

*Carmen-Veronica Bobeanu, et al.*

Modeling of Discrete Event Systems: A Holistic and Incremental Approach Using Petri Nets

http://portal.acm.org/ft_gateway.cfm?id=1029178&type=pdf