Laboratoire Virtuel pour la Pompe Biologique dans le Domaine Mésopélagique

Nicolas Duhail

RAPPORT DE STAGE

Laboratoire Virtuel pour la Pompe Biologique dans le Domaine Mésopélagique

Auteur :

Nicolas DUHAIL
Master Recherche Informatique – Université de Rennes 1
Élève Ingénieur – Telecom Bretagne

Encadrants :

Marc PARENTHOEN
Pascal REDOU
Lab-STICC – IHSEV
Centre Européen de Réalité Virtuelle

Résumé

L'objectif du stage est de concevoir un laboratoire virtuel pour permettre l'expérimentation en réalité virtuelle de modèles de la pompe biologique dans le domaine mésopélagique. Actuellement, il est établi que l'interaction entre les particules qui sédimentent et les organismes de la zone mésopélagique (100-1000m) détermine l'efficacité de la pompe biologique qui varie spatialement et temporellement à différentes échelles ; mais il n'y a pas de consensus sur les mécanismes la contrôlant.

Ce document étudie quels concepts et outils facilitent l'étude et la compréhension des phénomènes impliqués dans le devenir du carbone produit à la surface des océans qui sédimente après avoir traversé différents réseaux trophiques entre le voisinage de la surface et les grandes profondeurs. Deux verrous informatiques principaux sont identifiés pour élaborer ce laboratoire virtuel : l'intrication de modèles multidisciplinaires et les interactions multi-échelles.

L'analyse de la dynamique de ce système complexe grâce à la théorie de compétition pour les ressources nous permet d'interpréter les interactions écologiques et biochimiques dans des régions stables de l'océan. Cet aspect ne prenant en compte que les interactions entre populations, nous avons étudié la pompe biologique sous une approche basée agent. L'écosystème virtuel créé s'est avéré stable et donc particulièrement intéressant pour la prévision de la réponse à des changements de paramètres. Cependant, ce genre de simulation basé agent est difficile à mettre en place et est lourd en calcul.

Une approche multi-échelles (basée sur la théorie du bilan énergétique dynamique) nous a orientés vers le laboratoire que nous avons réalisé, programmé en langage C. Une approche par paquets de particules nous a permis d'obtenir des résultats qui semblent cohérents dans un temps raisonnable.

Mots clés : laboratoire virtuel, modélisation, simulation, dynamique des systèmes, individu centré, multi-échelles, DEB, pompe biologique, cycle du carbone.

Table des matières

Introduction

Présentation du laboratoire

Le Centre Européen de Réalité Virtuelle (CERV) est un centre de recherche pluridisciplinaire inter-établissements rattaché à l'Ecole Nationale d'Ingénieurs de Brest (ENIB). Il accueille deux équipes de recherche : l'équipe IHSEV (Interaction Humain Système et Environnement Virtuel) du Lac-STICC et l'équipe du CREAD (Centre de Recherche sur l'Éducation, les Apprentissages et la Didactique) [1].

Créé à l'initiative du Laboratoire d'Ingénierie Informatique (LI2) de l'ENIB, le CERV rassemble des enseignants-chercheurs de nombreuses disciplines, telles que l'informatique, les mathématiques, l'éducation, l'acoustique, et la psychologie ergonomique. Le CERV a plusieurs utilités, puisqu'il intègre dans un même lieu :

- un centre de recherche inter-établissements pour fédérer les travaux de différentes équipes de recherche en privilégiant la vocation pluridisciplinaire de la réalité virtuelle ;
- un centre de transfert de technologie pour favoriser les relations recherche-entreprise autour de projets innovants utilisant la réalité virtuelle ;
- un centre de formation à la réalité virtuelle ;
- un centre de découverte de la réalité virtuelle pour sensibiliser les scolaires et le grand public aux sciences et technologies du futur.

Les recherches du CERV sont centrées sur la réalité virtuelle, les comportements autonomes, les environnements intelligents, l'interaction Humain-Machine et la modélisation et simulation de systèmes complexes. Le centre couvre un large spectre de thèmes de recherche, dont :

- la simulation interactive et la modélisation d'activités collaboratives ;
- l'analyse de l'interaction et des activités humaines ;
- les simulations phénoménologiques pour laboratoire virtuel ;
- la parallélisation de simulations individus-centrées à grande échelle ;
- la modélisation et validation de simulations individus centrées ;
- la recherche fondamentale : autonomie, épistémologie.

Présentation de la mission

Ce stage M2R informatique est réalisé au CERV, au sein de l'équipe IHSEV du Lab-STICC. Il s'inscrit dans le cadre d'une action soutenue par l'axe 2 du Labex Mer : complexité et efficacité de la pompe biologique. Il s'agit d'une collaboration entre trois laboratoires CNRS situés sur le Technopôle Brest-Iroise : LEMAR, LPO, et Lab-STICC.

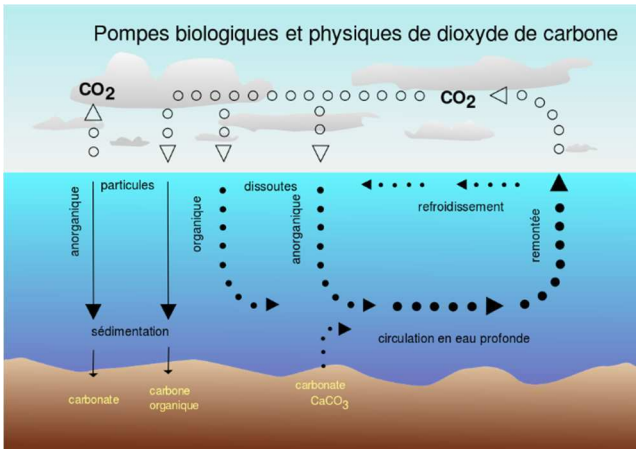
L'objectif du stage est de concevoir un laboratoire virtuel pour permettre l'expérimentation en réalité virtuelle de modèles de la pompe biologique dans le domaine mésopélagique proposé par les biogéophysiciens du LEMAR et du LPO. Actuellement, il est établi que l'interaction entre les particules

qui sédimentent et les organismes de la zone mésopélagique (100-1000m) détermine l'efficacité de la pompe biologique qui varie spatialement et temporellement à différentes échelles ; mais il n'y a pas de consensus sur les mécanismes la contrôlant. Ce stage étudie quels concepts et outils facilitent l'étude et la compréhension des phénomènes impliqués dans le devenir du carbone produit à la surface des océans qui sédimente après avoir traversé différents réseaux trophiques entre le voisinage de la surface et les grandes profondeurs.

Deux verrous informatiques principaux sont identifiés pour élaborer ce laboratoire virtuel : l'intrication de modèles multidisciplinaires et les interactions multi-échelles.

- Au niveau des simulations multi-modèles : la pompe biologique est un système complexe mêlant des phénomènes physiques, chimiques, biologiques et éthologiques. La posture épistémologique adoptée pour ce stage consiste à appliquer le paradigme de l'enaction aux modèles de la simulation. Chaque modèle est vu comme une entité autonome caractérisée par ses interactions. Nous nous efforçons d'identifier le caractère générique des modèles et des expérimentations menées en réalité virtuelle afin de faciliter l'évolution du laboratoire virtuel co-élaboré entre informaticiens et biogéophysiciens.
- Au niveau des interactions multi-échelles : en fonction des types de modèles impliqués dans les simulations (basé populations, particules, stochastique...) nous serons amenés à apporter des preuves mathématiques ou expérimentales sur les capacités de ces systèmes à représenter des interactions simultanément à différentes échelles et/ou à concevoir des algorithmes efficaces pour implémenter de telles simulations multi-échelles sur des architectures multicores.

En effet, la pompe biologique est un système complexe qui fait intervenir plusieurs phénomènes : à la surface, le phytoplancton fixe le carbone, puis sédimente vers le fond de l'océan, en s'agrégeant et se dissolvant en fonction de la microbiologie associée à ces particules. Elles sont de plus en interaction avec le zooplancton et le système trophique de la zone mésopélagique. Les phénomènes physico-chimiques de l'océan sont également à considérer : turbulence, température, concentrations chimiques. Il y aura donc un aspect continu d'une part pour les phénomènes physiques et un aspect agent d'autre part pour le comportement des individus du système trophique [2].



Cela nous amène à étudier les différentes échelles de ce système. Au niveau spatial, cela va donc de quelques micromètres (taille du phytoplancton) jusqu'à un kilomètre environ (colonne d'eau). Au niveau temporel, il y a un aspect continu (sédimentation, dissolution) et un aspect discret (désagrégation/agrégation du sédiment, broutage par le plancton). Aussi les échelles varient de quelques années pour une expérience virtuelle où l'on veut mesurer ce qui arrive au fond de l'océan en fonction du choix des modèles, à quelques millisecondes pour le broutage.

Le présent document est organisé de la façon suivante : dans un premier temps nous verrons comment aborder cette problématique en utilisant des systèmes d'équations différentielles, et plus spécifiquement en s'intéressant à la théorie de compétition pour les ressources. Ensuite nous étudierons le problème sous son aspect multi-échelles, tout d'abord grâce à une approche individu centrée, puis

nous aborderons la pompe biologique en utilisant le modèle DEB (bilan énergétique dynamique). Enfin nous présenterons le travail qui a été effectué durant ce stage.

I/ Approche par systèmes d'équations différentielles

Afin de réaliser un laboratoire virtuel pour des simulations de modèles de la pompe biologique, nous avons tout d'abord voulu comprendre les différents modèles auxquels nous serions confrontés. La première idée pour modéliser un système est d'utiliser un ou plusieurs système(s) d'équations différentielles (modélisation dynamique). Nous allons voir si cela s'avère concluant dans notre cas.

1- Définition des paramètres du modèle

La modélisation dynamique est la modélisation d'un système de manière continue. Elle est basée sur l'écriture d'équations différentielles qui régissent le système. Pour les résoudre à tout instant, il s'agit de discrétiser le système. La simulation de ce genre de modèles continus peut mener à des temps d'exécution très longs. Dans certains cas, il est possible de les réduire en se focalisant sur les sous-systèmes ayant les plus grands niveaux d'activité [3].

Comme nous avons pu le constater, la pompe biologique est un système complexe faisant intervenir de nombreuses entités différentes. Il est donc nécessaire afin d'obtenir une modélisation correcte d'effectuer en quelque sorte un inventaire des différents paramètres qui vont entrer en jeu ici.

Bien entendu, il est possible de faire différents choix de modèles : on peut par exemple se contenter de modéliser les phénomènes intervenant sur une simple particule de carbone (voir partie IV), ou bien appliquer différentes théories avancées par des géophysiciens.

a) *Resource Competition Theory*

Dans le domaine auquel on s'intéresse, la théorie de la compétition pour les ressources [4 ; 5] fournit un schéma pour interpréter les relations entre les organismes et leur environnement. En biologie, on appelle compétition la rivalité entre les espèces vivantes pour l'accès aux ressources de leur milieu. Ce phénomène permet la régulation des espèces, par auto-inhibition ou prédation par exemple. On obtient alors des écosystèmes contenant une biodiversité bien spécifique.

Il est nécessaire qu'il existe dans le milieu au moins une ressource en quantité limitée utilisée par deux espèces pour qu'on parle de compétition [6]. Cette ressource peut par exemple être l'eau, la nourriture, ou même un territoire en particulier. Certaines espèces ne disposent pas des capacités requises pour se joindre à la compétition : dans ce cas, elles doivent s'adapter ou mourir. C'est le principe d'exclusion compétitive.

Cette théorie de compétition va nous permettre d'étudier les relations entre l'écologie et la disponibilité des ressources. C'est une interprétation possible de la régulation des communautés de phytoplancton.

b) Paramétrage

En se basant sur la théorie précédemment introduite et en supposant que les mouvements physiques des organismes sont négligés, on peut écrire les équations suivantes :

$$\frac{dN}{dt} = -\mu_m \frac{N}{N + \kappa_N} P + S$$

$$\frac{dP}{dt} = \mu_m \frac{N}{N + \kappa_N} P - mP$$

P est un photoautotrophe alimentée par un macronutriment N approvisionné au système avec le taux S. μ_m est le taux de croissance maximal, il dépend de la lumière et de la température. Le paramètre κ_N représente la limitation en nutriment, et m prend en compte toutes les pertes comme le broutage par exemple.

Des paramètres supplémentaires vont être utilisés dans ce modèle, mais nous n'allons pas détailler leurs valeurs : un coefficient de saturation, un coefficient d'inhibition, la vitesse de sédimentation du phytoplancton, un intervalle de température… Certains de ces paramètres seront définis avec une valeur fixe, et d'autres auront une valeur choisie aléatoirement dans un intervalle donné.

2- Modélisation dynamique de la colonne d'eau

À partir des équations précédentes, dans des conditions parfaitement stables, lorsque le système a atteint son état d'équilibre [7], on a alors :

$$\bar{N} = \frac{\boldsymbol{\kappa_N m}}{\boldsymbol{\mu_m - m}} = \boldsymbol{R^*}$$

$$\bar{P} = \frac{S}{m}$$

On remarque que la concentration de la ressource limitante est déterminée par les caractéristiques de l'organisme. Il semble donc que R^* pourra être utile pour prédire l'issue de la compétition pour les nutriments uniquement dans certains environnements océaniques. On peut alors se demander dans quelles régions la théorie de compétition s'avère utile, et dans quelle mesure le phytoplancton régule les nutriments présents dans l'océan.

Pour répondre à ces questions, nous initialisons différents types de phytoplancton avec chacun des attributs physiologiques spécifiques, qui les rendent différents. Les paramètres de saturation en nutriment, de lumière et de température sont définis dans des intervalles de valeurs plausibles. Les interactions avec l'environnement, la compétition avec les autres phytoplanctons, et le broutage déterminent la composition des communautés de phytoplancton qui vont persister dans ce modèle. Notre écosystème virtuel de l'océan est ainsi régulé par des procédés similaires à ceux qui structurent les écosystèmes du monde réel. Ce modèle semble donc suffisamment complexe pour refléter les propriétés du phytoplancton et les interactions naturelles.

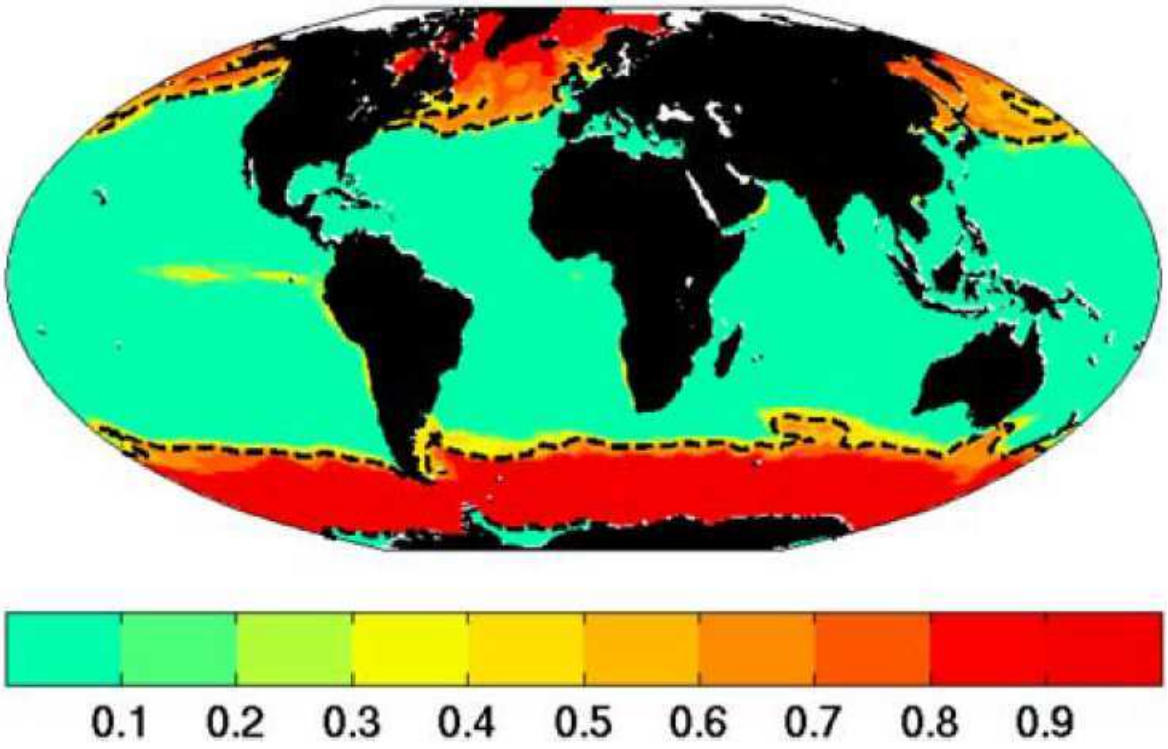
3- Expérimentations du modèle

Nous allons maintenant expérimenter ce modèle à travers deux configurations différentes. Dans la première, nous simplifions le modèle biogéochimique de l'océan en ne représentant qu'un unique macronutriment (nitrate ou phosphate par exemple). Dans la seconde, plus réaliste, plusieurs ressources sont présentes et on utilise d'autres types de phytoplancton.

a) Ressource unique

Dans cette configuration, la croissance dépend uniquement d'un macronutriment, le phosphate PO_4. De plus, on utilise uniquement deux classes de phytoplancton, une avec un faible ratio R^* (type 1), et l'autre avec une forte croissance (type 2). Chacune de ces classes possède la même sensibilité à la lumière concernant sa croissance, et la même probabilité d'être broutée par un prédateur. Ces deux classes de phytoplancton sont initialisées avec une distribution en masse identique, choisie relativement faible. Quant au champ de phosphate, il est initialisé à partir de valeurs liées à la climatologie [8].

On lance alors la simulation de la vie de ces organismes dans l'océan pendant des dizaines d'années. Il s'avère que les résultats évoluent très peu après une dizaine d'années, donc nous allons considérer cela comme étant un état quasi stable.



Cette figure représente la fraction de biomasse des phytoplanctons de type 2 (forte croissance) par rapport au total. Les lignes pointillées représentent la séparation entre les types 1 et 2 (50% de chaque). On remarque que le phytoplancton à croissance rapide domine la distribution en biomasse dans les hautes latitudes.

Le rapport R^* peut toujours être exprimé (avec quelques suppositions pour simplifier) en combinant la physiologie du phytoplancton et les pertes [9]. Cependant, ce rapport n'est plus linéaire,

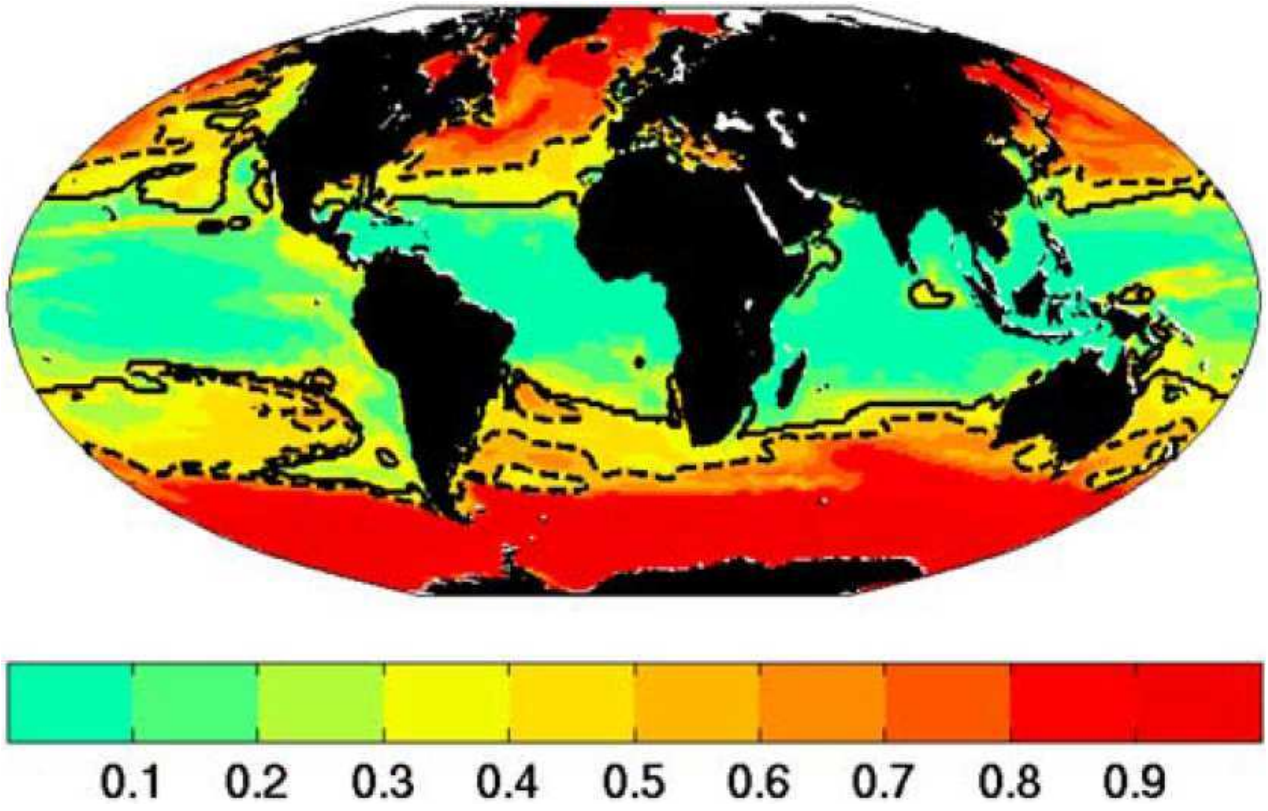
cela signifie donc que R^* dépend maintenant d'autres aspects du phytoplancton ainsi que de la production de nutriment (source).

La théorie de compétition pour les ressources suggère que les organismes avec le plus faible R^* vont surpasser tous les autres lorsqu'il qu'il n'y a qu'une seule ressource limitante, et sans autres facteurs limitants, ils seront même écartés. Comme attendu, ce formalisme se révèle efficace dans les environnements physiques relativement stables où il existe un équilibre entre croissance et mortalité.

b) Multiples ressources

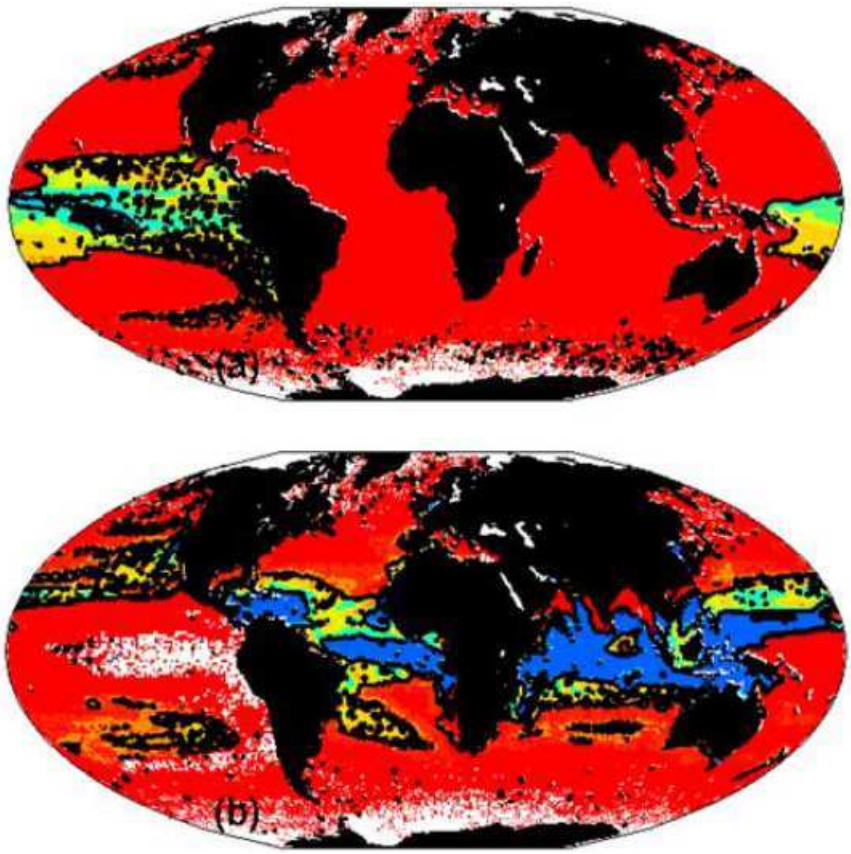
Nous allons désormais examiner les prévisions de cette théorie sur un système plus complexe, à savoir une simulation plus proche de la réalité. De nouveaux nutriments sont introduits : plusieurs composés nitrogènes, phosphore, fer et silice. De plus, la sensibilité à la lumière est maintenant prise en compte, ainsi que de nombreuses physiologies de phytoplancton supplémentaires.

Un total de 10 simulations a été effectué, chacune avec des propriétés physiologiques aléatoires, et des conditions initiales identiques. La durée de simulation est là encore de 10 années. Pour cette configuration, on donnera les résultats en moyenne sur l'ensemble des 10 tests.



Cette figure représente la fraction de biomasse des phytoplanctons à forte croissance (regroupés avec leurs analogues) par rapport au total. Les lignes pointillées représentent la séparation à hauteur de 50%. Les lignes continues indiquent l'intervalle où les organismes ayant le R^* le plus faible dominent. Tout comme dans le cas d'une ressource limitante unique, la biomasse à haute latitude est dominée par les phytoplanctons à croissance rapide alors qu'à basse latitude, les phytoplanctons ayant le rapport R^* le plus faible dominent la distribution. Cependant, on remarque que la transition entre ces régimes extrêmes se fait de manière beaucoup plus douce dans la configuration avec ressources multiples.

Le calcul du rapport R* est dorénavant bien plus compliqué, car il faut calculer un ratio différent par rapport à chaque ressource potentiellement limitante.



La figure (a) représente la proportion de fer par rapport au R^*_{min} tandis que la figure (b) fait de même pour les composés nitrogènes (NO_3+NO_2). La couleur rouge indique un excès de nutriments, et à l'inverse la couleur bleue indique une carence en nutriments. Les résultats obtenus précédemment correspondent à ces distributions : en effet, les phytoplanctons avec un faible ratio R^* ne sont pas en mesure d'utiliser le nitrate, la domination en biomasse de ces organismes se situe dans la zone bleue sur la figure (b), là où le nitrate est en quantité insuffisante [10].

Le rapport R^* est donc un outil approprié pour mesurer la compétitivité des espèces dans des régions spécifiques et dans un modèle relativement complexe de l'écosystème océanique. Dans les environnements les plus stables, il prévoie non seulement l'issue de la compétition entre phytoplanctons, mais indique également la concentration du nutriment limitant de manière quantitative. Nous avons vu que des organismes avec des ratios R^* assez proches pouvaient coexister [7]. La théorie de compétition pour les ressources fournit donc un formalisme intéressant pour interpréter les interactions écologiques et biochimiques dans des régions stables de l'océan. Cependant, nous n'avons vu ici que l'aspect population du phytoplancton. Nous allons y ajouter dans la partie suivante son aspect individu.

II/ Approche multi-échelles

Afin de modéliser un système complexe tel que l'océan (ou une partie de l'océan), les modèles continus (dynamiques) ne vont plus être adaptés. En effet, dans la communauté océanographique, il a été montré que ces modèles ne sont pas suffisants pour représenter les divers flux de particules au fond de l'océan [2]. De plus, les modèles développés jusqu'alors ne sont pas capables de relater la complexité de l'écosystème. Chaque population d'individus est représentée comme un seul groupe et non comme un ensemble de plusieurs individus, donc de nombreuses interactions entre individus d'un même groupe sont négligées. Pour surmonter ces difficultés, il est nécessaire de prendre en compte les comportements des populations mais aussi des individus.

1- Modélisation centrée individu

Après avoir appréhendé l'approche par équations différentielles qui est basée sur la modélisation centrée population, nous allons nous concentrer dans cette partie sur la modélisation centrée individu, aussi appelée modélisation basée agent.

a) Systèmes multi-agents

Un système multi-agents (SMA) est un système composé d'un ensemble d'agents, situés dans un certain environnement et interagissant selon certaines relations. Un agent est une entité caractérisée par le fait qu'elle est, au moins partiellement, autonome. Largement utilisés en intelligence artificielle, les SMA permettent de modéliser de manière intéressante des sociétés diverses et variées [11 ; 12 ; 13].

Lors de la création d'un SMA, on peut relever cinq problématiques principales :

- La problématique de l'action : comment un ensemble d'agents peut agir de manière simultanée dans un environnement partagé, et comment cet environnement interagit en retour avec les agents ? On peut également se demander quelle est la représentation de l'environnement par les agents, et comment ils collaborent dans cet environnement.
- La problématique de l'agent et de sa relation au monde : c'est le modèle cognitif dont dispose l'agent. Il doit être capable d'agir pour répondre au mieux à ses objectifs. Cette capacité à la décision est liée aux perceptions, aux représentations et aux croyances de l'agent.
- La problématique de la nature des interactions : on s'intéresse ici aux moyens d'interactions (quel langage ?) mais aussi à l'analyse et à la conception des formes d'interaction entre agents. Les notions de collaboration et de coopération sont centrales.
- La problématique de l'adaptation : il existe l'adaptation individuelle (appelée apprentissage) et l'adaptation collective (appelée évolution).
- La problématique de l'implémentation des SMA : on structure les langages de programmation en plusieurs types, du langage de formalisation et de spécification jusqu'au langage d'implémentation effective.

Ces problématiques vont nous permettre de spécifier des éléments d'architecture d'un SMA. Tout d'abord, les agents doivent être dotés d'un système de décisions et de planification à plusieurs.

Cela leur permettra par exemple de rechercher le meilleur chemin possible entre un point A et un point B.

De plus, les agents doivent être dotés d'un modèle cognitif. On peut citer par exemple le modèle BDI (*Beliefs – Desires – Intentions*). Il considère d'une part l'ensemble de croyances de l'agent sur son environnement, qui sont le résultat de ses connaissances et de ses perceptions, et d'autre part un ensemble d'objectifs. En croisant ces deux ensembles, on obtient un nouvel ensemble d'intentions qui peuvent ensuite se traduire directement en actions.

Ensuite, les agents doivent posséder un système de communication. Plusieurs langages spécifiques ont vu le jour à cet effet.

La question de l'adaptation reste très compliquée à ce jour. On peut citer l'exemple des virus (informatiques et biologiques) qui sont capables de s'adapter à leur environnement en mutant.

Nous allons voir dans le paragraphe suivant comment modéliser la colonne d'eau sous forme de système multi-agents.

b) Modélisation centrée individu de la colonne d'eau

Ce type de modélisation se base sur l'étude du comportement de chaque individu dans le but d'obtenir par intégration les propriétés de la population de plancton. Un modèle consiste en un certain nombre de groupes fonctionnels créés par l'utilisateur [14]. Un groupe fonctionnel est un ensemble d'individus (planctons) qui se comportent de la même manière. Leurs propriétés peuvent varier séparément (masse, volume, position, …) et il peut aussi y avoir des valeurs constantes pour tous les individus d'un même groupe.

On crée ensuite un ensemble de fonctions pour chaque groupe fonctionnel. Chacune de ces fonctions décrit un aspect du comportement de l'individu en réponse à ses propriétés biologiques et aux propriétés de son environnement local (voisinage). Cela permet de prendre en compte les variations intra-population, ce qui n'était pas le cas dans la modélisation dynamique (basée population).

Un agent possède deux attributs : il se comporte comme un unique plancton et il transporte une information concernant une sous-population dynamique de la population de planctons [15]. Sa physiologie va être modélisée par les fonctions suivantes : respiration, photosynthèse, prédation, satiété, digestion, excrétion, reminéralisation, reproduction, mort naturelle, … Le comportement de l'agent peut aussi être défini par des fonctions. Par exemple, une fonction permettant d'éviter ses prédateurs, ou encore une autre permettant de trouver un partenaire pour la reproduction.

La colonne d'eau, quant à elle, va être définie par ses concentrations en nutriments à chaque pas de discrétisation, sa température locale, ses turbulences…

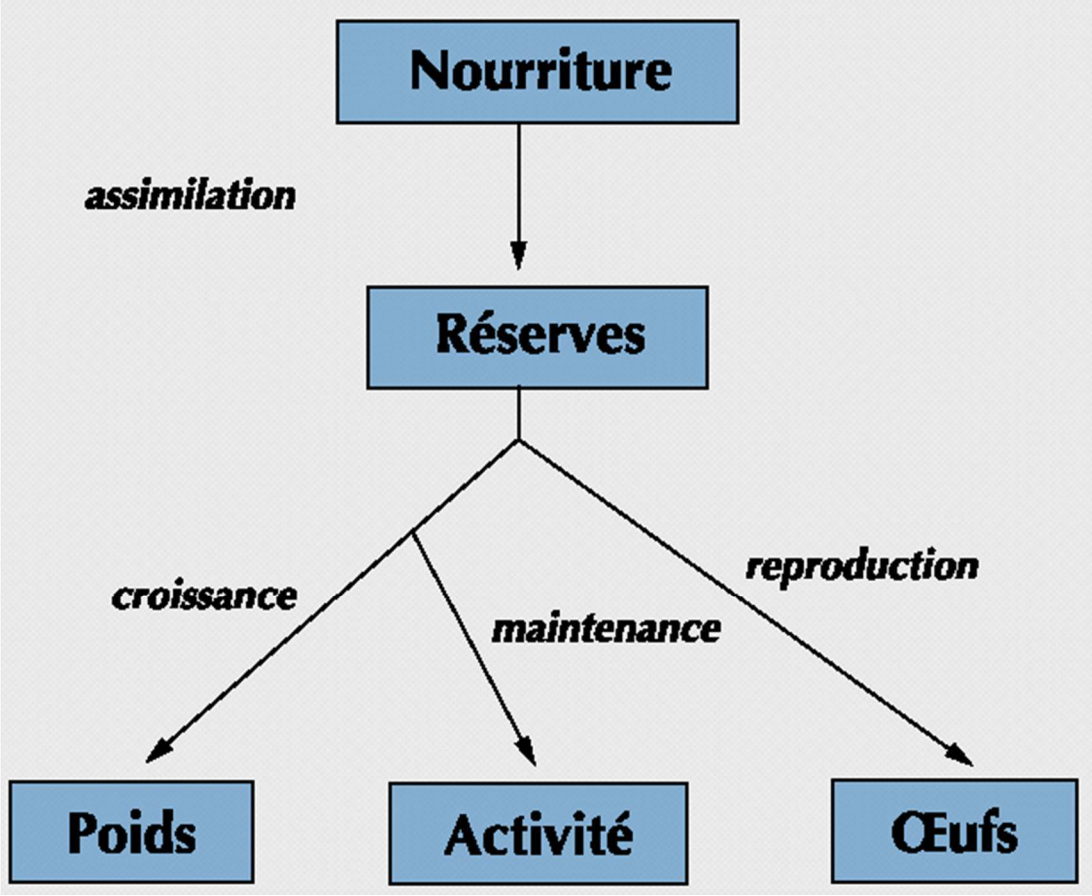
En utilisant ce modèle, il a été montré que l'écosystème virtuel ainsi créé est indépendant des conditions initiales, et donc stable [16]. Des études de sensibilité parcourant tout l'intervalle des paramètres et des niveaux de ressources ont démontré que cet écosystème est globalement stable. Cela est très intéressant pour prévoir la réponse de l'écosystème à un changement climatique par exemple.

La plupart des systèmes réels incluent des interactions parmi une grande variété de phénomènes physiques. De plus, l'échelle de temps et de taille de chaque processus peut varier de manière importante. Ainsi, les simulations numériques de ces problèmes multi-échelles nécessitent des modèles sophistiqués et des méthodes pour leur intégration, mais aussi des algorithmes efficaces et des techniques avancées de calcul. C'est pour ces raisons que nous allons présenter et utiliser une théorie différente dans les paragraphes suivants.

2- Modèle DEB

Le modèle DEB [17], ou bilan énergétique dynamique, est utilisé pour obtenir une description détaillée de la répartition et de l'utilisation de l'énergie dans les organismes. Il modélise le rôle de la température et de la concentration en nourriture (phytoplancton) sur cette répartition. Le potentiel de la théorie du bilan énergétique dynamique pour simuler le cycle de vie d'un organisme a été démontré à de nombreuses reprises. Ce modèle permet également des comparaisons entre espèces. Cependant, son applicabilité nécessite l'estimation de paramètres qui ne sont pas faciles à obtenir par des observations directes.

Le postulat est que les processus physiologiques sont les mêmes, quel que soit l'organisme considéré. Celui-ci distribue l'énergie qu'il absorbe en se nourrissant (en fonction de la température et à une vitesse proportionnelle à sa surface) entre deux « compartiments » : d'un côté la croissance, de l'autre la reproduction. Une fraction de cette énergie (proportionnelle au volume de l'organisme) est réservée à la maintenance des fonctions vitales.



La théorie DEB repose sur les fondements suivants [18] :

- conservation de la masse, de l'énergie et du temps ;
- relations entre volume et surface extérieure ;
- la production possède des contraintes stœchiométriques ;
- les substrats de l'environnement sont d'abord convertis en réserves avant d'être utilisés ;
- …

Les réserves sont un mélange de nombreux composants, donc on suppose qu'elles ne changent pas de composition. Cela permet d'avoir un état relativement stable. On peut se demander l'utilité d'inclure des réserves dans ce modèle. Une des explications est la mémoire métabolique des organismes : si la nourriture se fait rare, la croissance et la reproduction ne vont être affectées qu'après un certain délai. En effet, même sans se nourrir, la croissance continue un certain temps, et un embryon continue d'être alimenté par les réserves.

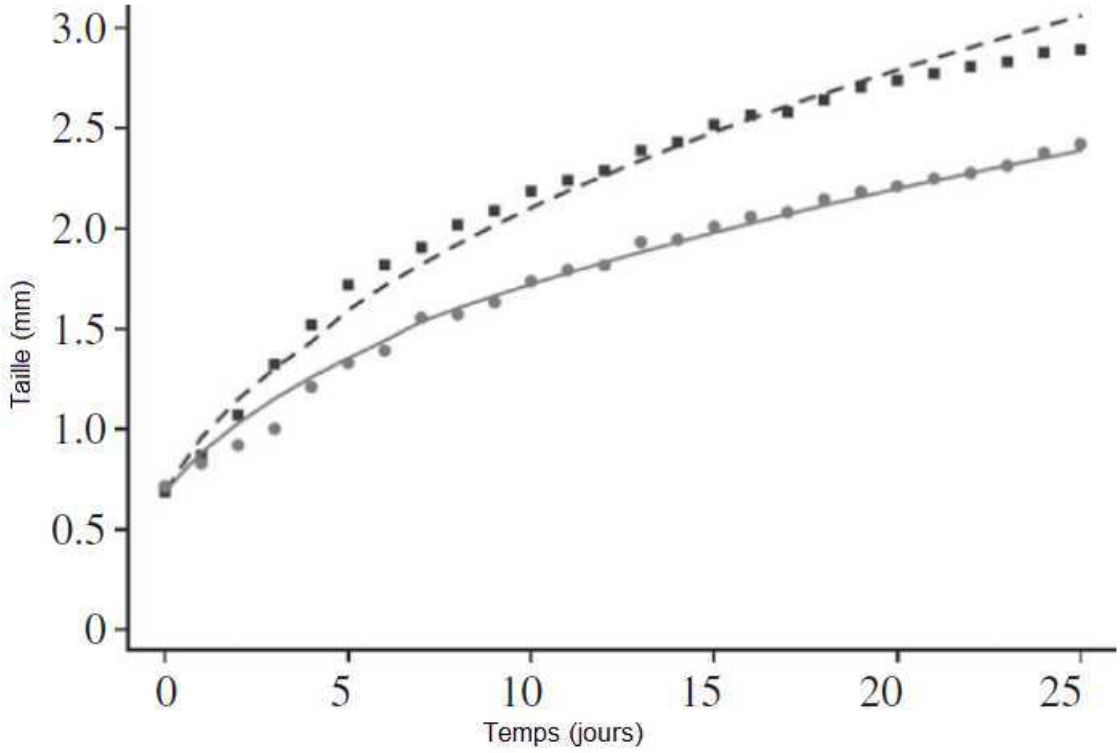
Il existe de nombreuses extensions du modèle DEB, utiles pour la simulation de modèles très complexes où le modèle standard ne s'avère pas assez détaillé (en médecine ou biologie moléculaire par exemple). Au contraire, dans certains domaines, on utilise des restrictions du modèle DEB pour se limiter aux sections pertinentes. Dans le modèle standard, les variables d'état de l'individu sont sa structure (masse) et ses réserves.

3- Application à la pompe biologique

La théorie DEB de base utilise plusieurs variables d'état pour caractériser un unique organisme. Cela devient alors techniquement compliqué de faire la transition vers la dynamique des populations. Les écologistes ayant besoin de modèles simples pouvant être utilisés en modélisation multi-échelles, nous allons utiliser dans cette partie une représentation simplifiée du modèle standard. En effet, une unique variable d'état va être nécessaire, la taille, et deux stages d'existence, jeune et adulte. Cette représentation contient assez d'informations sur la masse et le bilan énergétique pour obtenir des résultats corrects (par rapport aux données connues) sur la croissance, le vieillissement et la reproduction des individus en réponse à la disponibilité en nourriture [19].

Certains articles se sont concentrés sur les effets de la qualité de la nourriture (concentration des éléments pertinents), mais ici nous allons uniquement travailler sur la quantité de nourriture disponible (représentée sous forme de carbone). D'après plusieurs études expérimentales [20 ; 21 ; 22], les populations finissent par osciller autour d'une moyenne qu'on qualifiera d'équilibre. En utilisant des données (mesurées indépendamment) sur le rythme d'alimentation, l'assimilation, la mortalité et la respiration, la biomasse et la structure de la population à l'équilibre peuvent être prévues avec une assez bonne précision.

Nous allons de plus faire les suppositions nécessaires sur les individus pour obtenir une simplification majeure : le temps continu. Les individus sont catégorisés en tant que « jeunes » ou « adultes », et tous les individus d'une catégorie ont les mêmes taux physiologiques à chaque instant.



Cette figure représente la croissance d'un individu selon les suppositions faites précédemment. Les deux courbes correspondent à deux niveaux de nourriture différents. Les points sont les données expérimentales, et les courbes sont obtenues grâce à notre modèle simplifié de DEB. On constate que malgré la simplicité de notre modèle, la courbe de croissance s'accorde quasiment avec les données.

L'avantage de ce modèle simple est qu'il nous permet de passer à la dynamique des populations de manière directe. Nous pouvons directement calculer la démographie d'une population à l'équilibre. Avec des paramètres de croissance et de reproduction tirés de nos prévisions, et une estimation de l'impact de la dépendance en nourriture sur la mortalité des jeunes et des adultes, nous avons estimé la durée de la période « jeune » à environ 37 jours. Une estimation basée sur des données expérimentales pour cette intervalle de taille était entre 35 et 40 jours [22].

Un modèle de population basé sur ce bilan énergétique simplifié de la croissance, la reproduction et la mortalité est donc capable de capturer les propriétés essentielles de la dynamique comportementale de certains types de plancton, en présence de nourriture dite dynamique (croissance également).

Nous avons maintenant étudié différentes approches possibles de la pompe biologique : dynamique, basée agent, multi-échelles. Cela nous a guidés pour réaliser le travail présenté dans la partie suivante.

III/ Travail réalisé

Après avoir lu des articles concernant différentes approches de la pompe biologique, nous avons débuté la modélisation par son aspect continu. Pour cela, nous avons mis en place un modèle très simplifié où l'on s'intéresse simplement aux interactions entre particules.

1- Modèle analytique simple de la colonne d'eau

Dans ce modèle, nous allons uniquement prendre en compte les différents phénomènes physiques et chimiques qui s'appliquent à chaque particule. Dans un souci de simplicité, les interactions de la particule avec des prédateurs de type plancton ont été négligées dans ce modèle. Nous avons donc considérer la sédimentation, la reminéralisation, l'agrégation et la désagrégation.

On note φ la distribution des particules en suspension : φ(z,t,r) est le nombre de particules de rayon r, à l'instant t, à la hauteur z. Par ailleurs, on note ψ la fonction de grossissement.

Supposons maintenant que durant l'intervalle de temps δ_t, et indépendamment de z ou r :

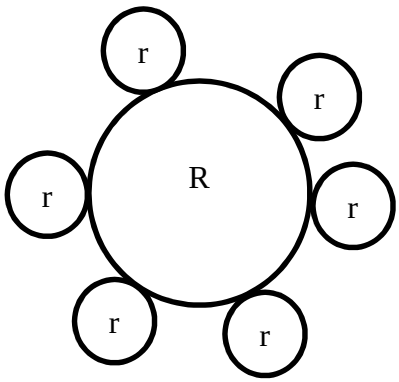
- Une proportion $\lambda_{sed}(\delta_t)$ de la population $\iiint \varphi(z,t,r)$ sédimente, sans reminéralisation ni agrégation ;
- Une proportion $\lambda_{rem}(\delta_t)$ de la population $\iiint \varphi(z,t,r)$ se reminéralise ;
- Une proportion $\lambda_{agr}(\delta_t)$ de la population $\iiint \varphi(z,t,r)$ s'agrège avec des particules de rayons inférieurs au sien ;
- Une proportion $\lambda_{des}(\delta_t)$ de la population $\iiint \varphi(z,t,r)$ se désagrège en particules de rayons plus petits.

On obtient donc la somme $\lambda_{sed}(\delta_t) + \lambda_{rem}(\delta_t) + \lambda_{agr}(\delta_t) + \lambda_{des}(\delta_t) = 1$.

Commençons par modéliser la fonction de grossissement : une particule de rayon R rencontre, durant l'intervalle de temps δ_t, $n(r,\delta_t)$ particules de rayon r inférieur à R. Ce nombre n de particules rencontrées peut être calculé grâce à la loi de Stokes par exemple si l'on suppose que l'on a un écoulement rampant, ou la loi de Newton sinon (cela dépend du nombre de Reynolds).

Loi de Stokes : $v = \frac{2r^2 g \Delta(\rho)}{9\eta}$

Supposons maintenant que les particules plus petites s'agrègent selon un modèle en étoile sur la grosse particule :



La particule de rayon R voit alors son rayon grossir d'une valeur de $\frac{2n(r,\delta_t)r^2}{\pi R}$.

Au total, la grosse particule verra son rayon prendre la nouvelle valeur suivante (fonction de grossissement) : $\psi(R,\delta_t) = R + \int_0^R \frac{2n(r,\delta_t)r^2}{\pi R} dr$. Afin de limiter les temps de calcul, on peut limiter la borne inférieure de l'intégrale à une valeur de R/10 par exemple, pour ne pas prendre en compte les particules trop petites par rapport à la particule considérée.

Si l'on suppose que les conditions (température, concentrations chimiques, turbulences, …) sont constantes dans toute la colonne d'eau, alors le facteur de reminéralisation (noté k) va être constant. Autrement dit, toutes les particules d'un certain rayon R à un instant donné et soumises uniquement à la reminéralisation auront un rayon r < R l'instant suivant (mais elles auront toutes ce même nouveau rayon). Ainsi, le terme dû à la reminéralisation intervenant dans φ(z,t,r) peut s'écrire :

$$\lambda_{rem}(\delta_t).\varphi(z - \delta_z(\delta_t, k.r), t - \delta_t, k.r)$$

Le terme $\delta_z(\delta_t,k.r)$ est la hauteur parcourue en moyenne par une particule de rayon k.r dans l'intervalle de temps δ_t (obtenue avec la loi de Stokes par exemple en connaissant la densité des particules et la viscosité du fluide).

La proportion due à l'agrégation (assemblage de petites particules qui forment une plus grosse) s'obtient simplement en intégrant sur l'ensemble des particules dont l'image du rayon par la fonction de grossissement est r (utilisation de la fonction indicatrice notée ici I). Quant à la désagrégation (fragmentation en petites particules) d'une particule de rayon R, on peut supposer qu'elle va faire accroître d'une quantité proportionnelle au rapport r / R la distribution des particules de rayon r.

On obtient ainsi l'équation globale de la distribution des particules de rayon r à une hauteur z et à l'instant t :

$$\begin{aligned}\varphi(z,t,r) = {} & \lambda_{sed}(\delta_t).\varphi(z - \delta_z(\delta_t, r), t - \delta_t, r) + \lambda_{rem}(\delta_t).\varphi(z - \delta_z(\delta_t, k.r), t - \delta_t, k.r) \\ & + \lambda_{agr}(\delta_t) \int_0^r \varphi(z - \delta_z(\delta_t, \rho), t - \delta_t, \rho).I_{[\psi(\rho,\delta_t)=r]}(\rho).d\rho \\ & + \lambda_{des}(\delta_t) \int_r^{+\infty} \frac{r}{\rho}\varphi(z - \delta_z(\delta_t, \rho), t - \delta_t, \rho).d\rho\end{aligned}$$

Malgré la simplicité apparente du modèle de base, et les hypothèses émises pour se simplifier la tâche, on se rend compte qu'il est très difficile de mettre en œuvre ce modèle et de l'appliquer numériquement à notre problème. C'est pourquoi nous avons assez vite abandonné la modélisation dynamique afin de se focaliser sur des modélisations basées agent ou multi-échelles.

2- Expérimentations en réalité virtuelle

Nous avons tout d'abord tenté de modéliser le comportement de chaque particule individuellement, mais au bout de quelques secondes on obtenait des ralentissements importants dus au nombre d'entités présentes dans la simulation. En effet, en lâchant quelques milliers de particules du haut de la colonne d'eau, on obtenait très vite des millions de particules en mouvement à cause de la désagrégation, et donc des millions de calculs à effectuer pour calculer les positions et les rayons de

toutes les particules à chaque instant. Bien entendu, les ordinateurs dont nous disposions n'étaient pas adaptés à ce type de calcul.

a) Paquets de particules

Pour pallier ce problème, nous avons alors décidé de modéliser les particules par paquets afin d'avoir moins d'entités à traiter. Ainsi, dans le code que nous avons choisi d'écrire en langage C pour obtenir de meilleures performances (proximité avec les instructions de l'ordinateur), une *Particle* sera composée de plusieurs particules, et l'on pourra agir directement sur ce nombre ou sur la masse lors des calculs.

```
struct _particle {
  // limites hautes et basse dans la colonne
  double zmin,zmax;
  // vitesses des particules
  double s;
  double Re; // nombre de Reynolds
  // informations nécessaires pour l'utilisation optimisée d'interactor
  int nbox,ind;
  // autres paramètres caractérisant la distribution des particules
  double r;
  double mass; // total mass transported by particle population
  double n; // number of particles in particle population
  double surface; // surface of the whole set of particles in particle population
  double relativeDensity;
  // informations nécessaires pour la sédimentation
  double lastSedimentTime;
  Activity sedimentation;
  // informations nécessaires pour la reminéralisation
  WaterColumn * wc;
  double lastRemineralizeTime;
  Activity remineralization;
  // informations nécessaires pour l'agrégation
  double lastAggregateTime;
  Activity aggregation;
};
```

b) Interacteur

Nous avons mis en place un *interactor* afin de faciliter les différentes interactions entre chaque entité. Dès la création d'une nouvelle entité (plancton ou paquet de particules par exemple), celle-ci est immédiatement enregistrée dans l'*interactor* avec les entités déjà existantes. Cela permet aux autres entités d'effectuer leurs actions sur toutes les entités présentes dans l'*interactor*, ou seulement celles présentes dans leur voisinage immédiat (voir fonction *initParticleApproxNeighborhoodIterator*).

```
void initInteractor(double zmin, double zmax, double dz){
  interactor.zmin=zmin;
  interactor.zmax=zmax;
  interactor.dz=dz;
  interactor.nboxes=(zmax-zmin)/dz+0.5;
  if (interactor.nboxes<0) interactor.nboxes=0;
  interactor.particleList=(ParticleList
*)malloc(interactor.nboxes*sizeof(ParticleList));
  for(int i=0;i<interactor.nboxes;i++) {
```

```
    interactor.particleList[i].particles=(Particle **)malloc(sizeof(Particle *));
    interactor.particleList[i].particles[0]=NULL;
    interactor.particleList[i].size=0;
    interactor.particleList[i].capacity=1;
  }
}

void addParticle(Particle * part){
  // part->nbox est l'index de la case où part est enregistré
  // part->ind est l'index de part dans la case
  int n=getParticleList(part);
  part->nbox=n;
  if ( (n>=0) && (n<interactor.nboxes) ) {
    ParticleList * pl=&(interactor.particleList[n]);
    if (pl->capacity==pl->size) {// il faut alors agrandir la liste pl sans changer
les index des particle* déjà enregistrées dans cette liste
      pl->particles=(Particle **)realloc(pl->particles,(pl-
>capacity*=2)*sizeof(Particle *));
    }
    // enfin, on ajoute part en incrémentant size
    pl->particles[part->ind=pl->size++]=part;
  }
  else {
    part->ind=-1;
  }
  // printf("particle %p entre [%g,%g], enregistrée en (%d,%d)\n",
(void*)part,part->zmin, part->zmax, part->nbox,part->ind);
}

int removeParticle(Particle * part){// retour = succès (1)
  int ok=0;
  if( (part->nbox>=0) && (part->nbox<interactor.nboxes) ) {
    ParticleList * pl= &(interactor.particleList[part->nbox]);
    if ( (part->ind>=0) && (part->ind<pl->size) && (pl->particles[part->ind]==part)
) {
      ok=1;
      // on remplace pl->particles[part->ind] par la dernière de la liste (pour
boucher le trou) et l'on décrémente pl->size
      pl->particles[part->ind]=pl->particles[--pl->size];
      pl->particles[part->ind]->ind=part->ind; // mise à jour de l'index pour la
particule déplacée
      // printf("particle %p déplacée à l'index %d\n",(void*)pl->particles[part-
>ind],pl->particles[part->ind]->ind);
    }
    else printf("removeParticle : part->ind=%d, non valide\n",part->ind);
  }
  return ok;
}

void initParticleApproxNeighborhoodIterator(double zmin, double zmax,
ParticleIterator * it){
  if (zmin<interactor.zmin) zmin=interactor.zmin;
  if (zmax>interactor.zmax) zmax=interactor.zmax;
  // recherche des index concernés
  it->nboxmin = (zmin-interactor.zmin)/interactor.dz - 0.5; // premier index
  it->nboxmax = (zmax-interactor.zmin)/interactor.dz + 0.5; // dernier index
  if (it->nboxmin < 0) it->nboxmin=0;
  if (it->nboxmax > interactor.nboxes) it->nboxmax = interactor.nboxes;
  // initialisation des index courants
  it->nbox=it->nboxmin;
  it->index=0;
}
```

```
Particle * iteratorNext(ParticleIterator * it){
  Particle * part=NULL;
  while (!part && it->nbox<it->nboxmax) { // tant qu'on a pas trouvé le prochain et
qu'il reste des boites à examiner
    ParticleList * pl=&(interactor.particleList[it->nbox]);
    if (it->index<pl->size) part=pl->particles[it->index++];
    else {
      it->index=0;
      it->nbox++;
    }
  }
  return part;
}
```

c) Ordonnanceur

D'autre part, nous avions également besoin d'un ordonnanceur pour pouvoir gérer les différentes activités (sédimentation, agrégation, reminéralisation, …) de chaque entité. Ce *scheduler* dispose d'une fonction pour ajouter une activité à exécuter à un instant donné, d'une fonction pour supprimer une activité, et d'une fonction qui renvoie la 1ère activité de la liste tout en la supprimant pour ne l'exécuter qu'une seule fois.

```
void pushActivity(Activity * act){
  // construction de la nouvelle cellule
  ActivityCell * cell=(ActivityCell *)malloc(sizeof(ActivityCell));
  cell->activity=act;
  // recherche de la position d'insertion
  ActivityCell * currentCell = scheduler.activities->head;
  ActivityCell * lastCell=NULL;
  while (currentCell && currentCell->activity->t <= act->t) {
    lastCell=currentCell;
    currentCell=currentCell->next;
  }
  // insertion de la cellule
  if(!lastCell){// insertion en tête
    scheduler.activities->head=cell;
    cell->next=currentCell;
  }
  else{// insertion entre lastCell et currentCell
    lastCell->next=cell;
    cell->next=currentCell;
  }
  scheduler.activities->n++;
}

void removeActivity(Activity * act){
  // recherche d'act dans le scheduler
  ActivityCell * currentCell=scheduler.activities->head;
  ActivityCell * lastCell=NULL;
  while (currentCell && currentCell->activity != act) {
    lastCell=currentCell;
    currentCell=currentCell->next;
  }
  if (currentCell) { // act a bien été trouvé dans currentCell
    if (lastCell) { // destruction de la cellule
      lastCell->next=currentCell->next;
      free(currentCell);
    }
    else { // destruction en tête
```

```
      scheduler.activities->head=currentCell->next;
      free(currentCell);
    }
    scheduler.activities->n--;
  }
}

Activity * popActivity(void){ // n'est appelé que si activityNumber()>0
  ActivityCell * head=scheduler.activities->head;
  Activity * activity=head->activity;
  scheduler.activities->head=head->next;
  scheduler.activities->n--;
  free(head); // libération de la cellule construite par pushActivity()
  return activity;
}
```

d) Gestion des paquets

Voici maintenant une partie du code permettant la gestion des particules. Dans la fonction *newParticle*, on commence par renseigner tous les attributs du paquet, puis on l'ajoute dans l'*interactor* et enfin on renseigne les activités du paquet que l'on ajoute dans le *scheduler*. On remarquera également la fonction *splitParticle* qui coupe un paquet de particules en deux lorsque celui-ci devient trop grand. La masse est alors divisée en deux de manière égale.

```
void updateParticleSpeed(Particle * part){
  // loi de Stokes (écoulement rampant)
  double deltaD=part->relativeDensity-1;
  if (deltaD<=0) printf("Warning : particle density lower than water\n");
  double r_lim=pow((3*81*eta*eta)/(2*rho*rho*g*deltaD),1.0/3); // rayon assurant
l'égalité des vitesses Stokes et Newton
  if (part->r<r_lim) {
    part->s=(2*part->r*part->r*g*rho*deltaD)/(9*eta); // formule de Stokes
    part->Re=2*part->s*part->r*rho/eta;
  }
  // loi de newton si vitesse plus grande (écoulement laminaire)
  //printf("Re_stokes(s=%g)=%g\n",part->s,part->Re);
  else {
    part->s=sqrt(6*g*part->r*deltaD);
    part->Re=2*part->s*part->r*rho/eta;
    //printf("\tRe_newton(s=%g)=%g\n",part->s,part->Re);
  }
  // printf("r_lim=%g, r=%g, Re=%g\n",r_lim,part->r,part->Re);
  // if (part->Re>1e5) printf("Re_newton(s=%g)=%g\n",part->s,part->Re);
}

Particle * newParticle(double zmin, double zmax, double t0, double r, double mass,
double relativeDensity){
  Particle * part=(Particle *)malloc(sizeof(Particle));
  part->zmin=zmin;
  part->zmax=zmax;
  part->r=r;
  part->mass=mass;
  part->relativeDensity=relativeDensity;
  part->n=part->mass*3/(4*M_PI*(part->r*part->r*part->r)*rho*relativeDensity);
  part->surface=part->n * 4*M_PI*part->r*part->r;
  updateParticleSpeed(part);
  part->wc=NULL;
  // enregistrement dans l'interactor
  // printf("newParticle : enregistrement dans l'interactor\n");
```

```
  addParticle(part); // nbox et ind sont alors renseignés
  // mise en place des activités
  // printf("newParticle : mise en place des activités\n");
  // sedimentation
  part->lastSedimentTime=t0;
  part->sedimentation.entity=part;
  part->sedimentation.t=t0;
  part->sedimentation.method=&sediment;
  pushActivity(&part->sedimentation);
  // remineralization
  part->lastRemineralizeTime=t0;
  part->remineralization.entity=part;
  part->remineralization.t=t0;
  part->remineralization.method=&remineralize;
  pushActivity(&part->remineralization);
  // aggregation
  part->lastAggregateTime=t0;
  part->aggregation.entity=part;
  part->aggregation.t=t0;
  part->aggregation.method=&aggregate;
  pushActivity(&part->aggregation);
  //printf("NewParticle at %p :\n",(void*)part);
  //printParticle(part);
  totalParticleNumber++;
  return part;
}

void destroyParticle(Particle * part){
  // enlever toutes ses activités
  removeActivity(&(part->sedimentation));
  removeActivity(&(part->remineralization));
  removeActivity(&(part->aggregation));
  // enlever de l'interacteur
  removeParticle(part);
  // enfin libérer la mémoire
  free(part);
  totalParticleNumber--;
}

Particle * splitParticle(Particle * part){
  // créer une particle en coupant part au milieu depuis sa position actuelle
  // cette méthode doit concerver la masse
  double z0=(part->zmin+part->zmax)/2;
  Particle * p=newParticle(z0,part->zmax,part->lastSedimentTime,part->r,part-
>mass/2,part->relativeDensity);
  setWaterColumn(p,part->wc);
  // mise à jour de la particle scindée
  part->zmax=z0;
  part->mass/=2;
  part->n/=2;
  part->surface/=2;
  return part;
}
```

e) Production primaire

Il est également nécessaire de mettre en place une production primaire, c'est-à-dire les populations de particules qui vont « apparaître » en haut de notre colonne d'eau. En réalité, cette production est créée à la surface de l'océan grâce à la lumière, mais nous ne nous occupons ici que de la vie de cette production primaire par la suite.

```
void produceParticle(void * partialPrimaryProduction, double t){
  PartialPrimaryProduction *
ppp=(PartialPrimaryProduction*)partialPrimaryProduction;
  Particle * part;
  double r=exp(ppp->pp->logr[ppp->index]);
  double dr=exp((ppp->pp->logr[ppp->pp->sizeR]-ppp->pp->logr[0])/(2*ppp->pp-
>sizeR));
  double r1=r/dr, r2=r*dr;
  // n= int_r1^r2 A.r^(-epsilon)dr * Volume
  double n=ppp->pp->A/(-ppp->pp->epsilon+1)*(pow(r2,-ppp->pp->epsilon+1)-pow(r1,-
ppp->pp->epsilon+1))*getParticleDeltaZmax();
  double relDens=(*(ppp->pp->relativeDensityFunctionOfSize))(r);
  double mass = n * 4*M_PI*r*r*r/3 * rho*relDens;
  ppp->pp->totalProductedMass+=mass;
  part=newParticle(ppp->pp->zmin, ppp->pp->zmin+getParticleDeltaZmax(), t,
              r, mass, relDens
              );
  setWaterColumn(part,ppp->pp->wc);
  // printf("PrimaryProduction[%d]. New particle :\n", ppp->index);
  // printParticle(part);
  // compute next activity time (when part->zmin is increased by particleDeltaZmax)
  double dt=getParticleDeltaZmax()/part->s;
  if ((ppp->act.t+=dt)<ppp->pp->t_end) {
    pushActivity(&(ppp->act));
  }
  else {
      double zmax=part->zmin+(ppp->pp->t_end-t)*part->s;
      double coef=zmax/part->zmax;
      part->zmax=zmax;
      ppp->pp->totalProductedMass-=part->mass;
      part->mass*=coef;
      ppp->pp->totalProductedMass+=part->mass;
      part->n*=coef;
      part->surface*=coef;
      updateParticle(part);
      printf("end of primary production[%d] : radius=%g\n", ppp->index, r);
      }
}

void  initPrimaryProduction(PrimaryProduction * pp, double t0, double t_end, double
zmin, double rmin, double rmax, int sizeR, double A0, double epsilon0, WaterColumn
* wc, double (*relativeDensityFunctionOfSize)(double) ) {
  pp->zmin=zmin;
  pp->rmin=rmin;
  pp->rmax=rmax;
  pp->sizeR=sizeR;
  pp->A=A0;
  pp->epsilon=epsilon0;
  pp->relativeDensityFunctionOfSize=relativeDensityFunctionOfSize;
  pp->wc=wc;
  pp->totalProductedMass=0;
  pp->t_end=t_end;
  pp->logr=(double*)malloc((sizeR+1)*sizeof(double));
  pp->logr[0]=log(rmin);
  double dlogr=(log(rmax)-pp->logr[0])/sizeR;
  for (int i=0;i<sizeR;i++) pp->logr[i+1]=pp->logr[i]+dlogr;
  pp-
>produceParticles=(PartialPrimaryProduction*)malloc((sizeR+1)*sizeof(PartialPrimary
Production));
  for (int i=0;i<=sizeR;i++){
    pp->produceParticles[i].pp=pp;
    pp->produceParticles[i].index=i;
    pp->produceParticles[i].act.entity=&(pp->produceParticles[i]);
```

```
    pp->produceParticles[i].act.t=t0; // next time is now
    pp->produceParticles[i].act.method=&produceParticle;
    pushActivity(&(pp->produceParticles[i].act));
  }
}
```

f) Activités des particules

Sédimentation

Observons maintenant les différentes activités que peut exécuter un paquet de particules. Pour la sédimentation, on doit juste mettre à jour la vitesse du paquet puis calculer sa nouvelle position. Ensuite, on doit remettre l'activité sédimentation dans le *scheduler* pour l'exécuter à nouveau plus tard.

```
void sediment(void * particle, double t){
  Particle * part=(Particle *)particle;
  double dt=t-part->lastSedimentTime;
  //printf("sediment : particle %p, dt=%g, t=%g, lastT=%g\n",(void*)part,dt,t,part-
>lastSedimentTime);
  part->lastSedimentTime=t;
  updateParticleSpeed(part);
  double dz=part->s*dt;
  part->zmin+=dz;
  part->zmax+=dz;
  // vérification de la limite mésopélagique
  if (part->zmin+part->zmax > 2*benthos.zmin) {
    //printf("Zone mésopélagique\n");
    benthos.method(part,&benthos);
    destroyParticle(part);
  }
  else {
    // mise à jour de l'activité si nécessaire
    double nextTime=t+0.5*(part->zmax-part->zmin)/part->s;
    //printf("Sedimentation %p nextime : %g (t=%g)\n", (void*)part, nextTime, t);
    part->sedimentation.t=nextTime;
    pushActivity(&part->sedimentation);
    // mise à jour auprès de l'interactor
    updateParticle(part);
  }
}
```

Reminéralisation

La reminéralisation nécessite quelques calculs supplémentaires pour définir la masse qui est dissoute dans l'océan. De la même manière que pour la sédimentation, on rajoute l'activité dans le *scheduler* si nécessaire.

```
void remineralize(void * particle, double t){
  Particle * part=(Particle *)particle;
  double dt=t-part->lastRemineralizeTime;
  part->lastRemineralizeTime=t;
  //printf("remineralize : particle %p, dt=%g, t=%g,
lastT=%g\n",(void*)part,dt,t,part->lastRemineralizeTime);
  // calcul des taux de reminéralisation
  // ce taux est proportionnel à la surface des particules
  // et dépend de la température et de la concentration dans la colonne d'eau
  double z0=(part->zmin+part->zmax)/2;
  double tmpture=temperature(z0);
```

```
  double massConcentration=concentration_mass(z0);
  double
tx=remineralizationRatePerTimeUnitPerSurfaceUnit(part,tmpture,massConcentration);
  int indexZ0=(z0-part->wc->zmin)/part->wc->dz;
  if (indexZ0<0 || indexZ0>=part->wc->size) {
    printf("Warning : remineralization indexZ0 inapproprié ; ");
    indexZ0 = indexZ0<0 ? 0 : part->wc->size-1;
    printf("Forcé à %d\n",indexZ0);
  }
  double * whereToExport=part->wc->exported_element+indexZ0;
  double massOneParticle=(4*M_PI*part->r*part->r*part->r/3)*part-
>relativeDensity*rho;
  double lostMass=tx*part->surface*dt;
  if (lostMass>part->mass-massOneParticle) {
      lostMass=part->mass;
      part->mass=0; // particle need to be destroyed
    }
  else part->mass-=lostMass;
  // la quantité réminéralisée est tranferée à la colonne en z0
  *whereToExport += lostMass;
  //printf("Remineralize : massPart=%g\n",massPart);
  if (part->mass) {
    // mise à jour des rayons des particles (part->n is constant)
    part->r=pow(3*part->mass/(part->n*4*M_PI*rho*part->relativeDensity),1.0/3); //
relativeDensity is suposed to be constant
    part->surface=part->n*4*M_PI*part->r*part->r;
    updateParticleSpeed(part);
    // mise à jour de l'activité
    double totalRemineral_dt=part->mass/(tx*part->surface); // temps nécessaire à
la disparition des particules de rayon r
    double nextHalfColumnBox_dt=0.1*(part->zmax-part->zmin)/part->s; // temps
nécessaire pour se déplacer d'une demi boite
    // double nextHalfColumnBox_dt=0.5*part->wc->dz/part->s; // temps nécessaire
pour se déplacer d'une demi elemnt water column
    double nextTime = t +
((totalRemineral_dt<nextHalfColumnBox_dt)?totalRemineral_dt:nextHalfColumnBox_dt);
    //printf("Remineralization %p nextime : %g (t=%g, totremdt=%g,
nexthalfbox=%g\n", (void*)part, nextTime, t, totalRemineral_dt,
nextHalfColumnBox_dt);
    part->remineralization.t=nextTime;
    pushActivity(&part->remineralization);
  }
  else {
    //printf("Reminéralisation complète de la particule %p\n",(void*)part);
    destroyParticle(part);
  }
}
```

Agrégation

L'activité d'agrégation est la plus complexe car on doit travailler sur plusieurs paquets de particules à la fois (ceux qui sont dans le voisinage du paquet auquel on applique l'activité). On commence par calculer la masse provenant d'un paquet du voisinage qui va venir s'agréger sur le paquet actuel puis on met à jour les différents paquets. Si la masse restante dans un paquet est nulle, il faut bien entendu le supprimer.

```
void aggregate(void * particle, double t){
  Particle * part=(Particle *)particle;
  double dt=t-part->lastAggregateTime;
  part->lastAggregateTime=t;
```

```
  //printf("aggregate : particle %p, dt=%g, t=%g,
lastT=%g\n",(void*)part,dt,t,part->lastAggregateTime);
  ParticleIterator pit;
  Particle * otherPart;
  ParticleList toBeDestroyed;
  // boucle de parcours des particules
  toBeDestroyed.particles=NULL;
  toBeDestroyed.size=0;
  toBeDestroyed.capacity=0;
  double minEstimDt=0;
  //if(badParticle(stdout, part)) { fprintf(stdout,"ERROR Aggregate() bad part
%p:\n",(void*)part);printParticle(stdout,part);}
  for (initParticleApproxNeighborhoodIterator(part->zmin, part->zmax, &pit);
       (otherPart=iteratorNext(&pit));){
    if(badParticle(stdout,otherPart)) { fprintf(stdout,"ERROR Aggregate() bad
otherPart %p:\n",(void*)otherPart);printParticle(stdout,otherPart);}
    // printf("aggregation : part %p avec other part
%p\n",(void*)part,(void*)otherPart);
    if (part->s > otherPart->s){ // c'est la particle qui va la plus vite qui
agrege
      if (part->zmax > otherPart->zmin && part->zmin < otherPart->zmax) { //
collision possible entre part et otherPart
      // grossissement d'une particule unique de part en dt
      // on récupère le volume commun
      double vcom=fmin(part->zmax,otherPart->zmax)-fmax(part->zmin,otherPart-
>zmin); // *1m²
      double stickyness=computeStickyness(part,otherPart);
      double tx;
      if (part->Re>1) tx=stickyness* 4*M_PI*part->r*part->r * part->s * otherPart-
>relativeDensity*rho * (1-otherPart->s/part->s); // turbulences de part accrétant
les others
      else tx=stickyness* 4*M_PI*otherPart->r*otherPart->r * part->s * otherPart-
>relativeDensity*rho * (1-otherPart->s/part->s); // trajetoire other rencontrant le
point d'arrêt (rampant)
      // tx est la quantité de matière agrégée par unité de temps
      // par une particule de part (en kg/s) provenant de otherPart
      double aggregedMass=(part->n*vcom/(part->zmax-part->zmin))*tx*dt;
      double massOneOtherPaticle=(4*M_PI*otherPart->r*otherPart->r*otherPart-
>r/3)*otherPart->relativeDensity*rho;
      if (!(aggregedMass>=0)) {
        fprintf(stdout,"ERROR : aggregate aggregedMass<0 !\n");
        fprintf(stdout,"to :\t");printParticle(stdout,part);
        fprintf(stdout,"from :\t");printParticle(stdout,otherPart);
        printf("---------\n");
      }
      if (aggregedMass > otherPart->mass-massOneOtherPaticle) {
        aggregedMass=otherPart->mass;
        otherPart->mass=0;
      }
      else otherPart->mass-=aggregedMass;
      // mise à jour de otherPart (à rayon const)
      otherPart->n=otherPart->mass*3/(4*M_PI*(otherPart->r*otherPart->r*otherPart-
>r)*rho*otherPart->relativeDensity);
      if (otherPart->mass) {
        double estimDt=otherPart->mass/tx;
        if (estimDt<minEstimDt) minEstimDt=estimDt;
      }
      else { // otherPart is added to the toBeDestroyed list
        if (toBeDestroyed.size>=toBeDestroyed.capacity) {
          if (toBeDestroyed.capacity)
toBeDestroyed.particles=(Particle**)realloc(toBeDestroyed.particles,((toBeDestroyed
.capacity*=2)*sizeof(Particle*)));
          else { // first time allocation
```

```
            toBeDestroyed.particles=(Particle**)malloc(sizeof(Particle*));
            toBeDestroyed.capacity=1;
          }
        }
        toBeDestroyed.particles[toBeDestroyed.size++]=otherPart;
      }
      // mise à jour de part (à nombre constant)
      // printf("%g mass aggreged\n",aggregedMass);
      double previous_mass=part->mass;
      part->mass+=aggregedMass;
      // relative density is set to the barycenter of part and other part density,
pondered by their respective masses
      // an other way for computing the resulting relative density could be used
      part->relativeDensity=(previous_mass*part-
>relativeDensity+aggregedMass*otherPart->relativeDensity)/part->mass;
      part->r=pow(3*part->mass/(part->n*4*M_PI*rho*part->relativeDensity),1.0/3);
      part->surface=part->n*4*M_PI*part->r*part->r;
      updateParticleSpeed(part);
      }
    }
  }
  // destruction effective des otherPart
  for (int i=0;i<toBeDestroyed.size;i++)
destroyParticle(toBeDestroyed.particles[i]);
  free(toBeDestroyed.particles);
  // mise à jour de l'activité d'aggregate
  double nextDt=0.1*(part->zmax-part->zmin)/part->s; // temps pour déplacement d'un
dixième de boite
  if (minEstimDt>0){
    if (nextDt>minEstimDt) nextDt=minEstimDt;
  }
  double nextTime = t + nextDt;
  // printf("aggregate : part.r=%g, part.s=%g, nextDt=%g, nextTime=%g\n",part->r,
part->s, nextDt, nextTime);
  part->aggregation.t=nextTime;
  pushActivity(&part->aggregation);
}
```

g) Plancton

Nous avons également programmé un modèle de plancton qui possède les actions de broutage et de défécation (création d'une nouvelle particule). Le plancton mange selon sa taille, puis dans la méthode de broutage, on ajoute l'activité de défécation après le temps de digestion nécessaire.

```
void graze(void * entity, double t){
  Plankton * plankton=(Plankton *)entity;
  double dt=t-plankton->lastGrazingTime;
  plankton->lastGrazingTime=t;
  // The plankton eats good size particles among its neibourhood
  ParticleIterator pit;
  Particle * part;
  ParticleList toBeAteThenDestroyed;
  // boucle de parcours des particules
  toBeAteThenDestroyed.particles=NULL;
  toBeAteThenDestroyed.size=0;
  toBeAteThenDestroyed.capacity=0;
  for (initParticleApproxNeighborhoodIterator(plankton->zmin, plankton->zmax,
&pit);
       (part=iteratorNext(&pit));){
```

```
    if (plankton->mouthSize<10*part->r && part->r < plankton->mouthSize){ // good
size
      if (plankton->zmax > part->zmin && plankton->zmin < part->zmax) { // good
place : plankton may eat part
      // toBeAteThenDestroyed will be ordered by decreasing radius
      // part is inserted at the right place
      if (toBeAteThenDestroyed.particles) {
        if(toBeAteThenDestroyed.size>=toBeAteThenDestroyed.capacity) { // need
realloc
          Particle **
particles=(Particle**)malloc((toBeAteThenDestroyed.capacity*=2)*sizeof(Particle*));
          // find the index for insertion
          int i=0;
          while(i<toBeAteThenDestroyed.size && toBeAteThenDestroyed.particles[i]->r
> part->r) {
            particles[i] = toBeAteThenDestroyed.particles[i];
            i++;
          }
          // insert at index i
          particles[i++]=part;
          // complete with others
          toBeAteThenDestroyed.size++;
          while(i<toBeAteThenDestroyed.size) {
            particles[i]= toBeAteThenDestroyed.particles[i-1];
            i++;
          }
          free(toBeAteThenDestroyed.particles);
          toBeAteThenDestroyed.particles=particles;
        }
        else {// insert without realloc
          // find the index for insertion
          int i=0;
          while(i<toBeAteThenDestroyed.size && toBeAteThenDestroyed.particles[i]->r
> part->r) i++;
          // move toward right index greater than i
          for (int j=toBeAteThenDestroyed.size++; j>i; j--)
toBeAteThenDestroyed.particles[j]=toBeAteThenDestroyed.particles[j-1];
          // insert part at index i
          toBeAteThenDestroyed.particles[i]=part;
        }
      }
      else {
        toBeAteThenDestroyed.particles=(Particle**)malloc(sizeof(Particle*));
        toBeAteThenDestroyed.capacity=1;
        toBeAteThenDestroyed.particles[toBeAteThenDestroyed.size++]=part;
      }
      }
    }
  }
  // grazable particle mass for an individual plankton
  //if(toBeAteThenDestroyed.size) printf("graze@%g : %d particles to
eat\n",t,toBeAteThenDestroyed.size);
  double grazable = (toBeAteThenDestroyed.size) ? plankton->grazingRate*dt : 0;
  if (grazable>(plankton->stomachSize-plankton->stomach)) grazable=plankton-
>stomachSize-plankton->stomach;
  // now planktons eat particles until grazable is reached for each plankton
  for (int i=0; i<toBeAteThenDestroyed.size; i++) {
    Particle * part=toBeAteThenDestroyed.particles[i];
    //if(badParticle(stdout, part)) { fprintf(stdout,"ERROR graze() bad part
%p:\n",(void*)part);printParticle(stdout,part);}
    // common volume
    double vcom=fmin(plankton->zmax,part->zmax)-fmax(plankton->zmin,part->zmin); //
*1m²
```

```
    // number of involved planktons
    double n_plankton=plankton->n*vcom/(plankton->zmax-plankton->zmin);
    // particle mass involved
    double mass_part=part->mass*vcom/(part->zmax-part->zmin);
    // effective mass of grazed particle
    double grazedMass=(n_plankton*grazable<mass_part) ? n_plankton*grazable :
mass_part;
    double grazedMassPerIndividual=grazedMass/plankton->n;
    //printf("%g planktons ate %g kg particles in %g s (%g kg/s per indiv.) with
radius %g m\n",plankton->n,grazedMass,dt,grazedMassPerIndividual/dt,part->r);
    // update stomach (repartition on each individual plankton)
    plankton->stomach+=grazedMassPerIndividual;
    grazable-=grazedMassPerIndividual;
    // update mass (destroy if null), number and surface for part;
    part->mass-=grazedMass;
    if(part->mass>0) {
      part->n=part->mass*3/(4*M_PI*(part->r*part->r*part->r)*rho*part-
>relativeDensity);
      part->surface=part->n * 4*M_PI*part->r*part->r;
    }
    else {
      printf("particle with radius %g completly ingested by plankton\n",part->r);
      destroyParticle(part);
    }
    if (!(grazable>0)) { // grazable is nul
      break;
    }
  }
  // wake up defecating if needed
  if(plankton->defecating.t<t && plankton->stomach>0) { // no poo activity but
stomach is no more empty
    plankton->defecating.t=t+plankton->digestionTime;
    pushActivity(&plankton->defecating);
  }
  // compute nextTime for grazing
  double nextDt=0;
  if (grazable>0) {
    nextDt=(plankton->stomachSize-plankton->stomach)/plankton->grazingRate;
    //printf("plankton is hungry @%g, estim next meal in %g s\n",t,nextDt);
  }
  else if (plankton->stomach==plankton->stomachSize){
    nextDt=plankton->digestionTime;
    //printf("plankton is full @%g, estim next meal in %g s\n",t,nextDt);
  }
  else {
    // printf("plankton needs food\n");
  }
  if (totalParticleNumber) { // grazing activity is going on only if some particle
still available in the waterColumn
  double estimDt=(plankton->zmax-plankton->zmin)/((2*plankton->mouthSize*plankton-
>mouthSize*g*rho*(2-1))/(9*eta));// vitesse de stokes pour des particules de la
taille de la bouche
  if (nextDt<estimDt) nextDt=estimDt;
  plankton->grazing.t=t+nextDt;
  pushActivity(&plankton->grazing);
  }
  // free useless allocated memory
  free(toBeAteThenDestroyed.particles);
}

void defecate(void * entity, double t){
  Plankton * plankton=(Plankton *)entity;
  double dt=t-plankton->lastDefecatingTime;
```

```
  plankton->lastDefecatingTime=t;
  double fecalPellet=plankton->digestionRate*dt;
  if (fecalPellet>plankton->stomach) {
    fecalPellet=plankton->stomach;
    plankton->stomach=0;
    //printf("hungry plankton %p at time %g\n",entity,t);
  }
  else plankton->stomach-=fecalPellet;
  if(fecalPellet){
    Particle * part=newParticle(plankton->zmax, plankton->zmax+plankton->zmax-
plankton->zmin, t, plankton->anusSize, plankton->n*fecalPellet, plankton-
>pelletRelativeDensity);
    setWaterColumn(part,plankton->wc);
    //printf("defecate new pellet @ %g\n",t); printParticle(stdout,part);
    //printf("defecate verifying good Particle...\n");
    //int bad=badParticle(stdout, part);
    //printf("bad=%d\n",bad);
    //if(bad) { fprintf(stdout,"ERROR defecate() bad part
%p:\n",(void*)part);printParticle(stdout,part);}
    //printf("defecate compute estimDt and nextDt...\n");
    double nextDt=(plankton->zmax-plankton->zmin)/part->s; // pellet time for
siking from zmin to zmax
    double estimDt=plankton->digestionTime;
    //printf("defecate estimDt=%g, nextDt=%g  @%g\n",estimDt,nextDt,t);
    if(estimDt>0) {
      if(estimDt>nextDt) nextDt=estimDt;
    }
    double nextTime = t + nextDt;
    //printf("defecate estim nextTime %g @%g\n",nextTime,t);
    plankton->defecating.t=nextTime;
    pushActivity(&plankton->defecating);
  }
  else {
    // need to graze for defecating again (defecating activity is updated by graze)
  }
  //printf("end of defecate @%g\n",t);
}
```

h) Outils de mesures

Enfin, pour effectuer les différentes mesures, on dispose d'un outil pour observer ce qui est arrivé au fond de l'océan à la fin de l'expérience (benthos), et d'un outil pour mesurer la masse exportée dans l'océan lors de la reminéralisation des particules.

```
double computeAndPrintBenthos(void){
  LogSubdivision * ls=(LogSubdivision *)benthos.data;
  double total_sedimented=0;
  double r=exp(ls->logrmin);
  double dr=exp((ls->logrmax-ls->logrmin)/ls->nb_r);
  printf("Sedimented into %g m :\n", benthos.zmin);
  for (int i=0; i<ls->nb_r; i++) {
    total_sedimented += ls->logSubdivision[i];
    printf("(%g) %g ",r,ls->logSubdivision[i]);
    r*=dr;
  }
  printf("(%g)\n-> total sedimented mass :\n%g",r,total_sedimented);
  return total_sedimented;
}

double computeAndPrintExported(const WaterColumn * wc){
```

```
  double total_exported=0;
  printf("Exported %s from %g m to %g m, with step %g m :\n", wc->name, wc->zmin,
wc->zmax, wc->dz);
  for (int i=0; i<wc->size; i++) {
    total_exported+=wc->exported_element[i];
    printf("%g ", wc->exported_element[i]);
  }
  printf("\n-> total %s exported :\n%g", wc->name, total_exported);
  return total_exported;
```

i) Résultats et perspectives

Les premiers tests de ces fichiers se sont révélés plutôt concluants : les résultats obtenus ne sont pas aberrants et correspondent parfois, selon les paramètres, à des mesures réelles. L'étape suivante sera d'enregistrer les traces du programme pour différents paramètres d'entrée (taille des paquets, discrétisation de la colonne d'eau, …) puis de comparer les résultats ainsi obtenus, et d'analyser quels sont les informations pertinentes ou au contraire incohérentes.

Conclusion

L'objectif du stage a été la conception d'un laboratoire virtuel pour l'étude de la pompe biologique dans le domaine mésopélagique. La modélisation des événements qui régissent ce phénomène a donc été à la base du travail demandé, puis la validation de ces modèles.

Il existe dans la littérature scientifique de nombreux ouvrages proposant des modélisations variées pour la pompe biologique. Après en avoir étudié plusieurs, nous nous en sommes inspirés pour réaliser notre propre modèle simplifié du cycle du carbone. Il a tout d'abord fallu écrire analytiquement toutes les équations régissant ce système complexe, puis nous avons programmé petit à petit un « laboratoire virtuel » permettant de simuler nos modèles.

Plusieurs ajustements ont dû être effectués pendant la période de programmation, le plus souvent à cause de temps de calcul beaucoup trop longs. Finalement, nous avons obtenu un outil sur lequel les particules sont rassemblées par paquets afin d'accélérer les calculs. Les planctons sont également intégrés à cette simulation, et les paramètres d'entrée sont modifiables.

Le fait d'avoir organisé les interactions particules/particules et planctons/particules sous forme d'activités permet d'en ajouter facilement si besoin est, si le modèle global vient à être modifié. En outre, des outils de mesure présents dans le code permettent de récupérer les résultats (masse de carbone exportée dans l'océan, masse sédimentée) dans des fichiers distincts afin de pouvoir les comparer plus tard.

Pour le moment, nous nous sommes concentrés sur les fonctionnalités du laboratoire virtuel, et nous n'avons pas encore analysé en profondeur les résultats obtenus selon divers paramètres d'entrée. Quand cela sera fait, il serait intéressant (si tout fonctionne comme attendu) de mettre en place un agent de type « caméra », qui suivrait le parcours d'une particule et montrerait à l'utilisateur les informations pertinentes de son parcours. Ainsi, l'évolution d'une particule (agrégation, reminéralisation, broutage

par un plancton) depuis le haut de la zone mésopélagique jusqu'au fond de l'océan pourrait être suivie en temps réel.

Références

[1] *Présentation du CERV*. (2013, Août 12). Récupéré sur Site Web du CERV: http://www.cerv.fr/

[2] Ragueneau, O. (2010). *Complex Ecosystems: a LAgrangian, VIrtual approach of the ocean carbon biological pump*. Plouzané, France.

[3] Muzy, A., & Zeigler, B. P. (2005). *Discrete event simulation of large-scale spatial continuous systems*.

[4] Tilman, D. (1977), *Resource competition between planktonic algae: An experimental and theoretical approach*, Ecology, 58, 338-348.

[5] Tilman, D. (1982), *Resource Competition and Community Structure*, *Monogr. in Pop. Biol.*, vol. 17, 296 pp., Princeton Univ. Press, Princeton, N. J.

[6] Begon, M.; Harper, J.L.; Townsend, C.R. (1996). *Ecology: Individuals, populations and communities*. 3rd edition. Blackwell Science, Oxford, U.K.

[7] Dutkiewicz, S., M. J. Follows, and J. G. Bragg (2009), *Modeling the coupling of ocean ecology and biogeochemistry*, Global Biogeochem. Cycles, 23.

[8] Conkright, M. E., H. E. Garcia, T. D. O'Brien, R. A. Locarnini, T. P. Boyer, C. Stephens, and J. I. Antonov (2002), *World Ocean Atlas 2001, vol. 4, Nutrients, in NOAA Atlas NESDIS*52, U.S. Govt. Print. Off., Washington, D. C.

[9] Wilson, J. B., E. Spijkerman, and J. Huisman (2007), *Is there really insufficient support for Tilman's R^* Concept? A comment on Miller et al.*, Am. Nat., 169, 700-706.

[10] Tozzi, S., O. Schofield, and P. Falkowski (2004), *Historical climate change and ocean turbulence as selective agents for two key phytoplanktonfunctional groups*, Mar. Ecol. Prog. Ser., 274, 123-132.

[11] Banas, N. S., Wang, D.-P., & Yen, J. (2003). *Experimental validation of an individual-based model for zooplankton swarming*.

[12] Grimm, V. (1999). *Ten years of individual-based modelling in ecology: what have we learned and what could we learn in the future?* Ecological Modelling 115, 129-148.

[13] Ramat, E., Preux, P., Seuront, L., & Lagadeuc, Y. (1998). *Modélisation multi-agents de systèmes naturels - Réflexions générales et application en biologie marine*.

[14] Hinsley, W., Field, A. J., and Woods, J. D. (2007). *Creating individual based models of the plankton ecosystem*. Lecture Notes in Computer Science, 4487: 111-118.

[15] Woods, J. D. (2005). *The Lagrangian Ensemble metamodel for simulating plankton ecosystems*. Progress in Oceanography 67: 84-159.

[16] Woods, J. D., Perilli, A., and Barkmann, W. (2005). *Stability and predictability of a virtual plankton ecosystem created with an individual-based model*. Progress in Oceanography, 67: 43-83.

[17] Kooijman SALM (2000). *Dynamic energy and mass budgets in biological systems*. Cambridge University Press, New York.

[18] Sousa, T., T. Domingos, J. C. Poggiale, and SALM Kooijman. (2010). *Dynamic energy budget theory restores coherence in biology*. Philosophical Transactions of the Royal Society B: Biological Sciences 365: 3413-3428.

[19] Nisbet, R. M., E. McCauley, and L. R. Johnson. (2010). *Dynamic energy budget theory and population ecology: lessons from Daphnia*. Philosophical Transactions of the Royal Society B: Biological Sciences 365: 3541-3552.

[20] Nisbet, R. M., McCauley, E., Gurney, W. S. C., Murdoch, W. W. & de Roos, A. M. (1997). *Simple representations of biomass dynamics in structured populations. In Case studies in mathematical modeling - ecology, physiology, and cell biology*, pp. 61-79. Upper Saddle River, NJ: Prentice Hall.

[21] Nelson, W. A., McCauley, E. & Wrona, F. J. (2005). *Stage-structured cycles promote genetic diversity in a predator–prey system of Daphnia and algae*. Nature 433: 413-417.

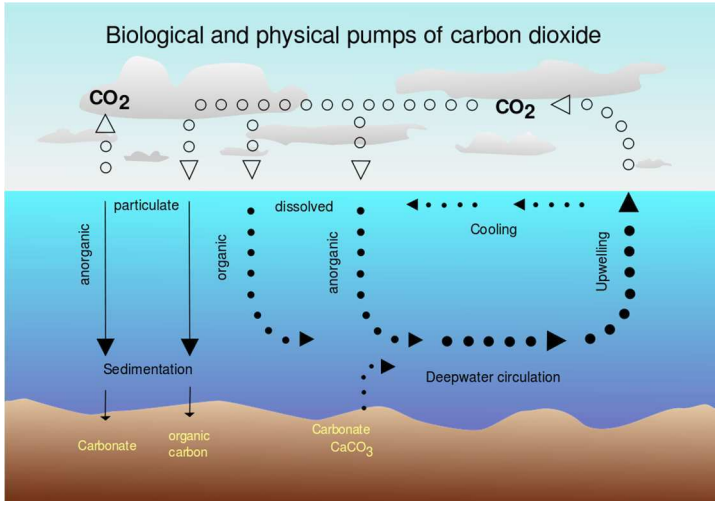
[22] McCauley, E., Nelson, W. A. & Nisbet, R. M. (2008). *Small-amplitude cycles emerge from stage-structured interactions in Daphnia–algal systems*. Nature 455: 1240-1243.

Annexe : Rapport Bibliographique

DEVS et ses extensions pour des simulations multi-modèles en interaction multi-échelles

Table des matières

Introduction



Dans le but d'étudier la pompe biologique (cycle marin du carbone) dans le domaine mésopélagique (zone de l'océan comprise entre 100m et 1000m de profondeur), ce qui sera le sujet de mon stage réalisé au CERV (Centre Européen de Réalité Virtuel), nous serons amenés à développer un laboratoire virtuel pour expérimenter des modèles des phénomènes impliqués dans cette colonne d'eau. Notre objectif est de pouvoir vérifier différents modèles fournis par des spécialistes du cycle marin du carbone. Ce laboratoire virtuel devra permettre de simuler un système multi-modèles interchangeables et modifiables facilement en tenant compte des interactions multi-échelles lors de leur simulation.

En effet, la pompe biologique est un système complexe qui fait intervenir plusieurs phénomènes : à la surface, le phytoplancton fixe le carbone, puis sédimente vers le fond de l'océan, en s'agrégeant et se dissolvant en fonction de la microbiologie associée à ces particules. Elles sont de plus en interaction avec le zooplancton et le système trophique de la zone mésopélagique. Les phénomènes physico-chimiques de l'océan sont également à considérer : turbulence, température, concentrations chimiques. Il y aura donc un aspect continu d'une part pour les phénomènes physiques et un aspect agent d'autre part pour le comportement des individus du système trophique [26].

Cela nous amène à étudier les différentes échelles de ce système. Au niveau spatial, cela va donc de quelques micromètres (taille du phytoplancton) jusqu'à un kilomètre environ (colonne d'eau). Au niveau temporel, il y a un aspect continu (sédimentation, dissolution) et un aspect discret (désagrégation/agrégation du sédiment, broutage par le plancton). Aussi les échelles varient de quelques années pour une expérience virtuelle où l'on veut mesurer ce qui arrive au fond de l'océan en fonction du choix des modèles, à quelques millisecondes pour le broutage.

Depuis son introduction en 1976 [38], le formalisme DEVS a donné lieu à de nombreux travaux pour formaliser de tels systèmes complexes. Nous examinerons parmi ces travaux ceux permettant la modélisation à la fois dynamique, basée agent, multi-échelles mais aussi permettant leur expérimentation. DEVS peut-il rendre compatible les aspects multi-modèles ? Notre problématique est de trouver quel formalisme DEVS permettrait la réalisation d'un laboratoire virtuel pour la pompe biologique dans le domaine mésopélagique.

Le présent document est organisé de la façon suivante : dans un premier temps nous étudierons un formalisme pour la modélisation, la simulation et l'analyse de systèmes complexes, appelé DEVS. La plupart des travaux de modélisation de systèmes complexes sont basés sur ce Framework. Ensuite nous analyserons les extensions de DEVS qui pourraient nous être utiles pour construire un laboratoire virtuel de la pompe biologique. Enfin nous aborderons quelques plateformes basées sur DEVS permettant la simulation de systèmes complexes.

I/ Formalisme DEVS

1- Présentation du Framework

Le formalisme DEVS (de l'anglais Discrete EVent system Specification) a été introduit par B.P. Zeigler en 1976 [38]. C'est un formalisme à événements discrets reconnu pour son expressivité, abstraction, universalité et son indépendance de toute implémentation. Il intègre intrinsèquement la dimension temporelle (elle est donc explicite) et permet de manipuler conjointement les deux concepts d'état et d'évènement. Il introduit également la possibilité d'évolution autonome du modèle grâce à la durée de vie des états.

C'est un formalisme modulaire et hiérarchique pour l'analyse, la modélisation et la simulation de systèmes complexes. Depuis sa création, il a été utilisé à de nombreuses reprises en ingénierie (systèmes de communications [21], …), en biologie [10] et en sociologie [19] par exemple. Sur la base de ce formalisme, de nombreux travaux de recherche ont donné lieu à des extensions, méthodes et approches [4 ; 22 ; 32].

Il existe deux objets élémentaires dans DEVS, un modèle atomique et un modèle couplé, qui échangent des informations sous la forme d'événements. Leurs caractéristiques sont décrites dans les deux parties suivantes.

2- Atomic DEVS

Un modèle atomique est défini comme un ensemble de ports d'entrée et de sortie, et un ensemble de fonctions :

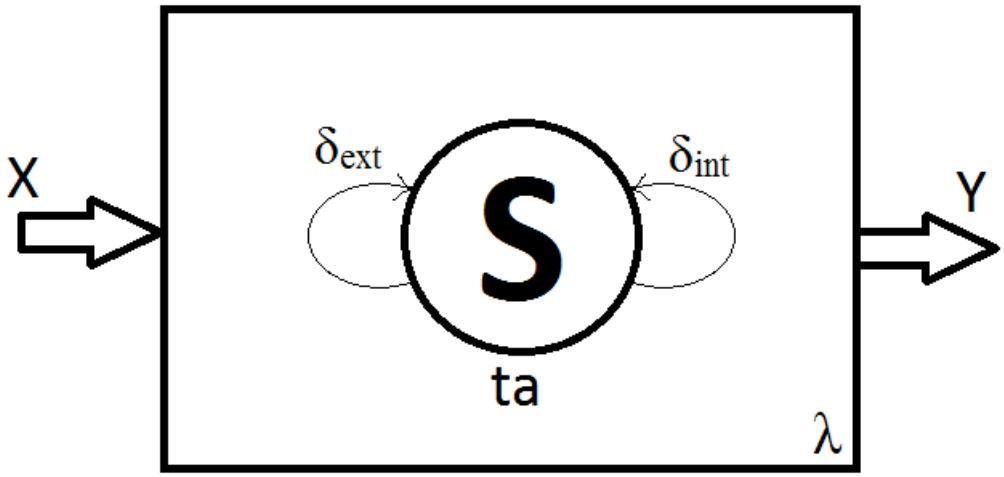
- la transition d'état interne ;
- la transition d'état externe ;
- une fonction qui peut générer des événements destinés à d'autres modèles ;
- une fonction qui gère la durée des différents états.

Ainsi, on peut définir un modèle atomique par un 7-uplet de la forme :

$$M =< X, Y, S, ta, \delta_{ext}, \delta_{int}, \lambda >$$

où X est l'ensemble des événements entrants, Y celui des événements sortants, S est l'ensemble des états du systèmes, ta est la fonction *time advance* qui gère la durée de vie d'un état, δ_{ext} est la transition d'état externe qui définit comment un événement d'entrée modifie l'état du système, δ_{int} est la transition d'état interne qui définit comment un état du système change sans action externe (quand le temps écoulé atteint la durée de vie de l'état), et λ est la fonction de sortie (elle définit comment un état du système génère un événement).

Voici un schéma représentant un modèle atomique DEVS symétrique :



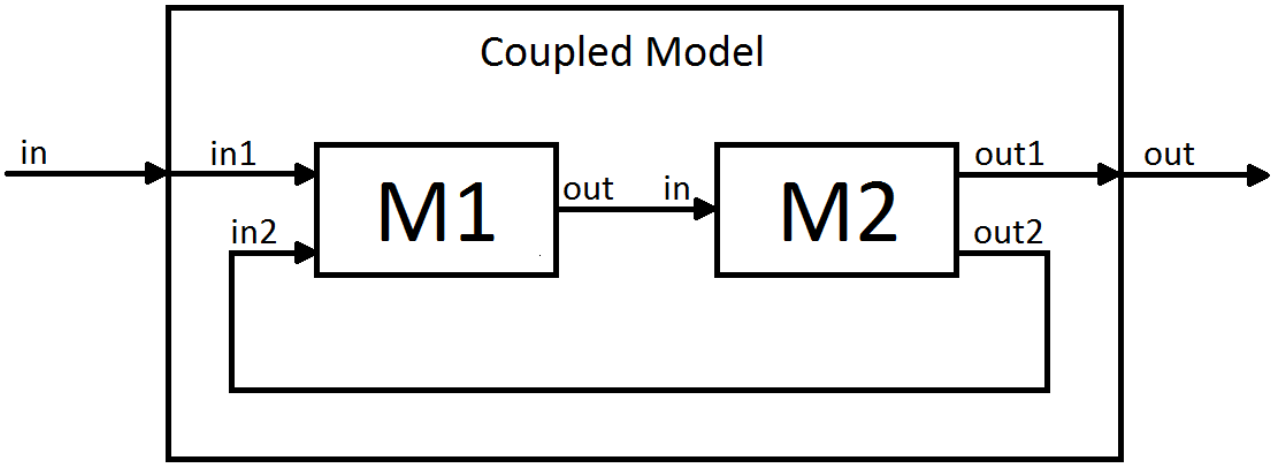
3- Coupled DEVS

Ce modèle décrit un système à événements discrets sous la forme d'un réseau de composants couplés. Les connexions entre différents modèles atomiques peuvent être faites par un modèle couplé. De la même manière que pour le modèle atomique, un modèle couplé est défini par un 8-uplet de la forme :

$$N = < X, Y, D, \{M_i\}, C_{xx}, C_{yx}, C_{yy}, Select >$$

où D est l'ensemble des noms des sous-composants, M_i est l'ensemble des sous-composants (pour chaque i, M_i est soit un modèle atomique soit un modèle couplé), C_{xx} est l'ensemble des couplages externes en entrée, C_{yx} est l'ensemble des couplages internes, C_{yy} est la fonction de couplage externe en sortie, et Select est une fonction qui définit quel événement choisir dans le cas d'événements simultanés (voir partie II-3 concernant l'optimisation pour la description d'un formalisme qui s'affranchit de cette fonction, Parallel-DEVS).

De plus, il est possible de rendre un modèle couplé équivalent à un modèle atomique quel que soit le système étudié [33]. Ci-dessous est présenté un schéma d'un modèle couplé DEVS :



Le formalisme DEVS de base est donc très utile pour formaliser des systèmes de toutes sortes grâce à l'apport de cohérence entre les modèles. Mais il possède des limites car il est trop générique et n'est pas suffisant pour la spécification de certains systèmes complexes [22 ; 32]. Ainsi, pour avancer

dans leurs travaux, des scientifiques ont proposé des variations de DEVS. Nous allons en aborder certaines dans la partie suivante.

II/ Extensions de DEVS

Depuis sa création en 1976, de nombreuses extensions du formalisme classique DEVS ont été développées, avec pour chacune des caractéristiques propres et des applications diverses. Nous verrons tout d'abord celles qui traitent de l'aspect multi-modèles, puis celles qui abordent l'aspect multi-échelles, et enfin celles qui considèrent l'optimisation.

1- Aspect multi-modèles

Certains formalismes basés sur DEVS traitent de la dynamique des systèmes, tandis que d'autres se concentrent plus particulièrement sur les agents. Quelques-uns essayent quant à eux de rassembler tout cela sous un seul et même formalisme, pour les systèmes hybrides.

a) Modélisation dynamique

La modélisation dynamique est la modélisation d'un système de manière continue. Elle est basée sur l'écriture d'équations différentielles qui régissent le système. Pour les résoudre à tout instant, il s'agit de discrétiser le système. La simulation de ce genre de modèles continus peut mener à des temps d'exécution très longs. On peut les réduire en se focalisant sur les sous-systèmes ayant les plus grands niveaux d'activité [24]. En utilisant DEVS, il a été montré qu'on peut réduire un programme complexe en une suite de fonctions simples [10].

Voici quelques formalismes basés sur DEVS pour la modélisation dynamique :

- dynDEVS [34] : un modèle dynDEVS est un ensemble de modèles DEVS plus une fonction de transition pour définir l'ordre des modèles [22]. Il est adapté aux modèles à structure dynamique.
- rho-DEVS [31] : cette extension introduit les ports dynamiques et les couplages multiples. Leur réunion autorise les modèles à activer ou désactiver certaines interactions en même temps.
- RT-DEVS (*Real Time* DEVS) [15] : propose une sémantique pour la spécification de systèmes en temps réel d'une façon modulaire.

Afin de modéliser un système complexe tel que l'océan (ou une partie de l'océan), les modèles continus ne vont plus être adaptés. En effet, dans la communauté océanographique, il a été montré que ces modèles ne sont pas suffisants pour représenter les divers flux de particules au fond de l'océan [26]. De plus, les modèles développés jusqu'alors ne sont pas capables de relater la complexité de l'écosystème. Chaque population d'individus est représentée comme un seul groupe et non comme un ensemble de plusieurs individus, donc de nombreuses interactions entre individus d'un même groupe sont négligées [26]. Pour surmonter ces difficultés, plusieurs approches sont actuellement développées, basées sur les agents.

b) Modélisation basée agent

Un système multi-agent (SMA) est un système composé d'un ensemble d'agents, situés dans un certain environnement et interagissant selon certaines relations. Un agent est une entité caractérisée par le fait qu'elle est, au moins partiellement, autonome. Largement utilisés en intelligence artificielle, les SMA permettent de modéliser de manière intéressante des sociétés diverses et variées [1 ; 11 ; 28].

Tout d'abord, les agents sont dotés d'un système de décisions et éventuellement de planification à plusieurs. On les dote ensuite d'un modèle cognitif : on peut par exemple citer le modèle BDI [3]. Enfin, les agents doivent posséder un système de communication.

Voici quelques formalismes basés sur DEVS pour la modélisation basée agent :

- *Cell*-DEVS [35] : cette extension permet la modélisation de systèmes physiques complexes en tant qu'espaces cellulaires.
- *Symbolic* DEVS [39] : ce formalisme se concentre sur le temps discret.

c) Modélisation hybride

Afin de prendre en compte à la fois des aspects système dynamique et des aspects agent, certains scientifiques se sont penchés sur les modèles hybrides. Lors de la modélisation de systèmes complexes, la complexité est généralement non seulement due au grand nombre de composants mais aussi à la diversité de ces composants et à leurs interactions.

Voici quelques formalismes basés sur DEVS pour la modélisation hybride :

- FD-DEVS *(Finite & Deterministic* DEVS) [16] : ce formalisme a été conçu pour l'analyse et la vérification de ses réseaux.
- SP-DEVS (*Schedule Preserving* DEVS) [18] : c'est une sous-classe de DEVS et FD-DEVS.
- G-DEVS (*Generalized* DEVS) [8] : spécialement imaginé pour les modèles hybrides.
- GK-DEVS (*Geometric & Kinematic* DEVS) [17] : cette extension permet la modélisation et la simulation de systèmes multi-composants 3D.

Après avoir vu l'utilisation de DEVS à travers l'aspect multi-modèles, intéressons-nous à présent à son utilisation dans la considération de l'aspect multi-échelles.

2- Aspect multi-échelles

En fonction des types de modèles impliqués dans les simulations, nous devrons déterminer si ces systèmes sont capables de représenter des interactions simultanément à différentes échelles, ou concevoir des algorithmes efficaces pour implémenter de telles simulations multi-échelles sur des architectures multicores.

La plupart des systèmes réels incluent des interactions parmi une grande variété de phénomènes physiques. De plus, l'échelle de temps et de taille de chaque processus peut varier de manière importante. Ainsi, les simulations numériques de ces problèmes multi-échelles nécessitent des modèles sophistiqués

et des méthodes pour leur intégration, mais aussi des algorithmes efficaces et des techniques avancées de calcul [25].

La conception de simulations multi-agents appliquées aux systèmes complexes pose entre autres le problème de la modélisation de comportements intervenant à des échelles spatiales, temporelles, et comportementales différentes, chacune pertinente pour représenter un des aspects du phénomène étudié. La simulation de systèmes complexes, qu'ils soient composés principalement d'agents réactifs comme en biologie cellulaire, ou plutôt d'agents cognitifs comme dans les sociétés artificielles, se doit de prendre en compte les divers niveaux d'organisation qui rendent compte des phénomènes étudiés. Aussi a-t-on vu apparaître des architectures destinées à représenter des emboîtements de systèmes et sous-systèmes (encapsulation d'agents ou d'environnements). Plus récemment, ce sont des problématiques d'appartenance multiple (à des groupes, à des espaces différents) qui ont pris le pas sur le simple problème d'encapsulation.

Il existe actuellement plusieurs plateformes de gestion d'échelles comme SWARM [23], AGRE [7], MASQ [29], IODA/JEDI [25]… Elles sont basées sur des formalismes comme DEVS, et nous allons voir dans cette partie deux exemples d'extensions de DEVS qui prennent en compte l'aspect multi-échelles.

a) Multi-Resolution DEVS

Le terme anglais *resolution* représente le degré de détail et de précision utilisé dans la représentation d'aspects du monde réel dans un modèle ou une simulation. Le principal but de la modélisation *multi-resolution* est d'extraire les détails de modèles complexes qui sont nécessaires pour restituer une partie du système étudié, et de se débarrasser des détails inutiles.

Le formalisme MR-DEVS, basé sur la structure dynamique de DEVS, a été créé pour les problèmes de modélisation *multi-resolution* [22]. Auparavant, il n'y avait rien qui permettait d'avoir des modèles cohérents entre eux pour ce genre de modélisation. MR-DEVS est un méta-modèle permettant de décrire des entités *multi-resolution*, c'est-à-dire des entités qui peuvent interagir avec des objets à une autre échelle que la leur.

Cette extension de DEVS a été introduite afin de modéliser la biologie au niveau système. Elle peut s'adapter pour contrôler les relations entre les niveaux micro et macro dans un système biologique. De plus, MR-DEVS peut modéliser un processus biologique à plusieurs niveaux hiérarchiquement et en terme de modularité [9].

Par rapport à un modèle classique DEVS (atomique ou couplé), un modèle MR-DEVS est définit par un 9-uplet comprenant des variables qui prennent en compte les changements de précision. Ces dernières peuvent impacter d'autres modèles MR-DEVS grâce à des fonctions de diffusion qui s'assurent de la cohérence du système modélisé.

b) Multi-Level DEVS

La base de la modélisation est l'abstraction. Ainsi, la réunion de différents niveaux d'abstraction est d'un intérêt primordial pour la communauté de modélisation et simulation depuis de nombreuses

années [32]. En biologie, nous allons donc avoir besoin d'un moyen de décrire les niveaux micro et macro et les liens qui existent entre ces niveaux.

Une nouvelle fois, une variation de DEVS va nous être utile, Multi-Level DEVS (ML-DEVS) [32]. Dans cette approche, le niveau macro peut accéder aux informations du niveau micro et vice-versa, les micro-modèles peuvent être activés simultanément par le macro-modèle, et les micro-modèles peuvent modifier la dynamique du niveau macro. Pour lier tout cela, on combine différentes méthodes, parmi lesquelles le couplage de valeurs et les activations synchrones [32].

Dans ML-DEVS, le modèle couplé est équipé d'un état et d'un comportement propre, afin que le niveau macro n'apparaisse pas en tant qu'unité à part. D'autre part, il faut définir comment chaque niveau affecte l'autre. Ces deux tâches sont liées. Pour propager l'information, l'idée de couplage de valeurs semble satisfaisante. Il s'agit de diffuser l'information du niveau macro sur des ports spécifiques, et les micro-modèles peuvent accéder aux macro-variables en définissant les bons noms de ports.

Au niveau micro et macro, les modèles sont toujours déclenchés par le temps et l'arrivée d'événements. Mais ici, on ajoute le fait que le niveau macro peut directement activer ses composants en leur envoyant un événement, et la dynamique du niveau macro peut être modifiée au niveau micro, si le nombre de composants dans un certain état dépasse un seuil fixé.

3- Optimisation

Afin de réduire les temps de calcul lors de simulation de systèmes complexes, des extensions de DEVS permettant la parallélisation des calculs ont été mises en place. Elles s'avèrent particulièrement efficaces pour la résolution d'équations différentielles [20] (modèle dynamique).

En se basant sur l'extension de DEVS nommée Parallel-DEVS (P-DEVS) [4], on va pouvoir utiliser des méthodes de simulations en parallèle. Les extensions traditionnelles utilisent des techniques où des branches de la hiérarchie de modèles sont réparties sur des calculateurs. Un point de synchronisation est alors nécessaire pour effectuer un test sur la causalité des événements. La méthode P-DEVS utilise une méthode simple de parallélisation des calculs en utilisant une parallélisation des modèles lorsqu'au dépilement d'une date dans l'échéancier, toutes les fonctions de transition internes ou externes des modèles sont effectuées en parallèle. La différence entre P-DEVS et DEVS classique est l'utilisation de sac d'événements (*bags*) en entrée sur la fonction de transition externe [4]. De plus, la fonction Select de DEVS classique est remplacée par une fonction de conflit, qui donne la possibilité au modélisateur de choisir, pour un modèle, entre ses événements externes et interne s'ils ont lieu à la même date.

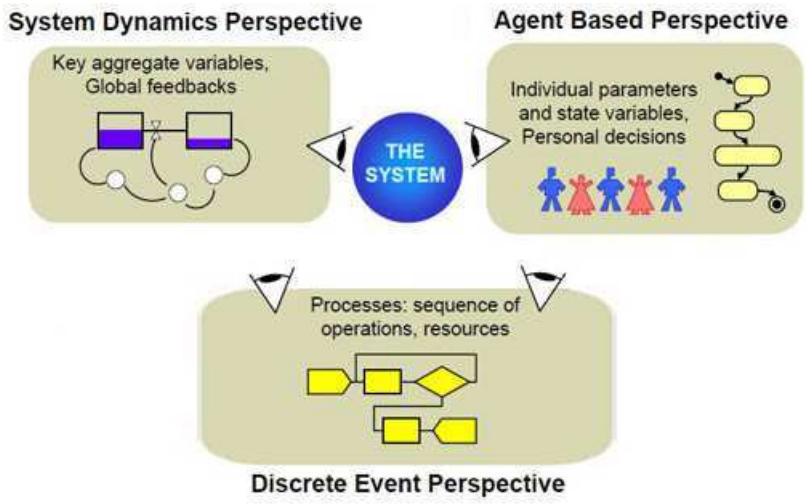
Les simulations parallèles se servent des algorithmes conservatifs pour garantir la cohérence et l'ordre causal des messages échangées entre des processus logiques. Elles utilisent des modèles modifiés de DEVS (comme P-DEVS) pour gérer la synchronisation [12 ; 40]. La simulation parallèle se fait en deux phases. Tout d'abord, tous les composants sont en lecture seule et ils lisent les informations sur les autres composants. À la fin de cette étape on synchronise pour vérifier la cohérence. Ensuite, tous les composants sont mis à jour selon le temps minimum d'un événement, puis resynchronise pour garantir la réussite de la mise à jour et la transmission des messages.

La performance de cet algorithme parallèle est principalement affectée par la répartition de la charge sur les composants, la communication entre composants, et la synchronisation entre processus logiques. Si le nombre de composants est largement supérieur au nombre de processeurs utilisés (ce qui est très souvent le cas en pratique), on peut s'attendre à obtenir des performances satisfaisantes [12].

Nous venons de passer en revue des formalismes divers permettant la spécification de modèles de systèmes complexes dans des cas variés. Il est maintenant nécessaire d'utiliser des outils pour la simulation de ces modèles, afin de pouvoir les valider ou non. C'est ce que nous allons aborder dans la partie suivante.

III/ Outils basés sur DEVS pour la simulation de systèmes complexes

Afin de simuler des systèmes complexes, plusieurs types de méthodes existent, à commencer par la simulation d'une société composée d'agents [1]. On peut également étudier la dynamique du système [24]. Notre objectif est de pouvoir facilement simuler plusieurs modèles d'un même système complexe grâce à des plateformes basées sur DEVS.



1- Expérimentations en réalité virtuelle

La réalité virtuelle est une simulation informatique interactive immersive, visuelle, sonore et/ou haptique, d'environnements réels ou imaginaires. Dans le cadre d'un laboratoire virtuel, elle permet aux scientifiques de réaliser des expériences *in virtuo* de modèles numériques de systèmes complexes [5]. Ce type d'expérimentations (dynamiques et interactives) est en effet nécessaire quand on modélise des systèmes comportant une structure et une dynamique complexes. Elles exploitent l'informatique et des interfaces comportementales en vue de simuler dans un monde virtuel le comportement d'entités qui interagissent en temps réel (entre elles et avec l'utilisateur).

En plus de pouvoir observer l'activité du modèle numérique en cours d'exécution sur un ordinateur, l'utilisateur peut tester la réactivité et l'adaptabilité du modèle en fonctionnement, profitant ainsi du caractère comportemental des modèles numériques. Une expérimentation *in virtuo* est ainsi une

expérimentation conduite dans un univers virtuel de modèles numériques en interaction et auquel l'homme participe. Une telle simulation participative en réalité virtuelle met en œuvre des modèles de types différents (multi-modèles) à des échelles spatio-temporelles différentes (multi-échelles) issus de domaines d'expertise différents. Elle est souvent complexe car son comportement global dépend autant du comportement des modèles eux-mêmes que des interactions entre modèles [30].

Un outil d'expérimentation conçu par le CERV pour étudier de tels modèles est le *virtuoscope* [30]. Ce néologisme désigne un laboratoire virtuel pour l'étude des systèmes complexes en s'appuyant sur les concepts, les modèles et les outils de la réalité virtuelle. Ce paradigme aide à formuler des modèles mixtes dans des échelles de temps et d'espaces différentes, ce qui est un prérequis pour l'étude de systèmes biologiques complexes [22 ; 26 ; 32].

2- Plateformes de simulation et laboratoires virtuels

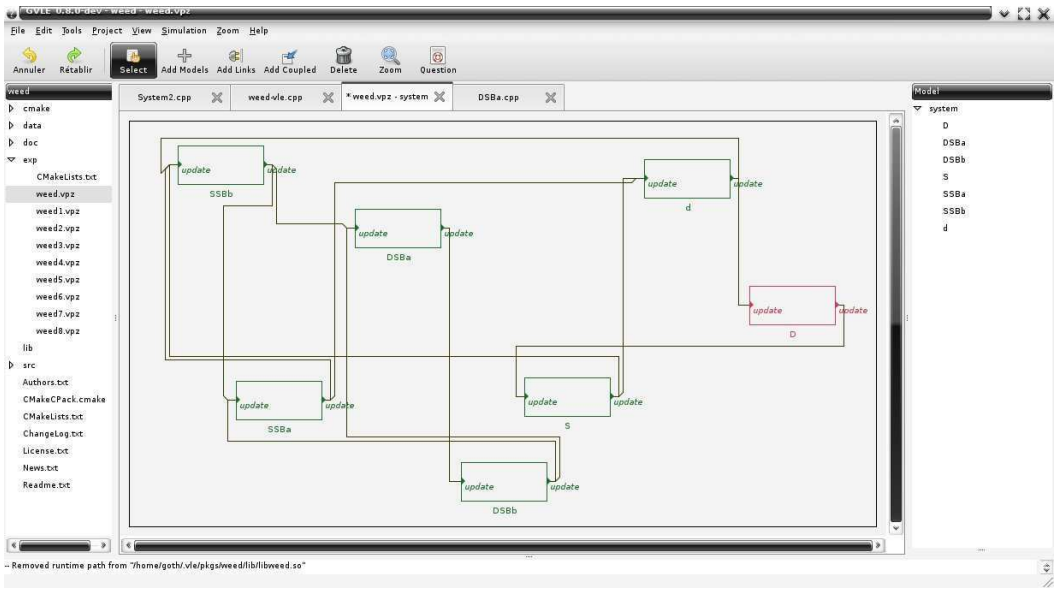
Le but d'un laboratoire virtuel est de pouvoir expérimenter un modèle plutôt qu'un autre. On rencontre le plus souvent dans le domaine de la simulation deux types principaux de laboratoires virtuels :

- Un laboratoire virtuel basé agent est utilisé pour la simulation de modèles de sociétés composés d'agents et utilise pour cela des technologies d'intelligence artificielle [6]. Ainsi, dans ce type d'environnement, chaque élément est modélisé par un agent, et on les place dans un environnement virtuel (sur ordinateur) pour observer l'évolution du système. On peut ensuite transposer les résultats de la simulation obtenus à des agents réels. Un laboratoire virtuel permet d'observer des événements qu'on ne pourrait pas voir en réalité [6]. Des laboratoires virtuels sont également utilisés pour acquérir des connaissances [6 ; 14] (destinés notamment à des élèves).
- Une autre manière de simuler des systèmes complexes est d'utiliser un logiciel de dynamique des systèmes. En effet, la dynamique des systèmes prend en compte les boucles de rétroaction internes et les effets retard qui affectent le comportement global du système. Elle est fondée sur des modèles qui sont une formalisation de nos suppositions à propos d'un système [13]. En dynamique des systèmes, faire tourner une simulation consiste à résoudre les équations mathématiques pour obtenir la valeur de chaque variable au cours du temps.

a) VLE

VLE, pour *Virtual Laboratory Environment*, est une plate-forme de multi-modélisation et de simulation de systèmes dynamiques basée sur le formalisme à événements discrets DEVS [27]. VLE fournit de nombreux programmes comme un simulateur, une interface utilisateur pour modéliser et développer des systèmes, et des outils pour analyser et visualiser les résultats de simulations. Les librairies fournies permettent également le développement de programmes personnalisés.

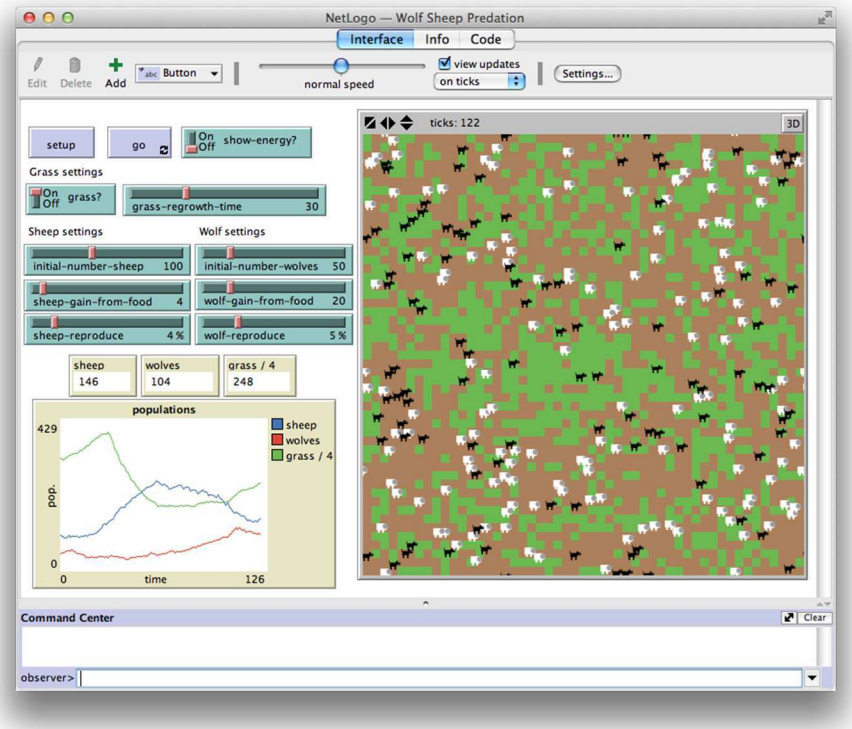
VLE rend possible la spécification de systèmes complexes spatiaux en terme d'objets et d'agents réactifs [27]. On peut citer le projet RECORD qui utilise VLE pour modéliser et simuler des agroécosystèmes [2].



b) NetLogo

NetLogo est un environnement de modélisation et de simulation multi-agents permettant d'observer la relation entre le comportement de chaque individu (niveau micro) et la structure générale qui s'en dégage [36] (niveau macro).

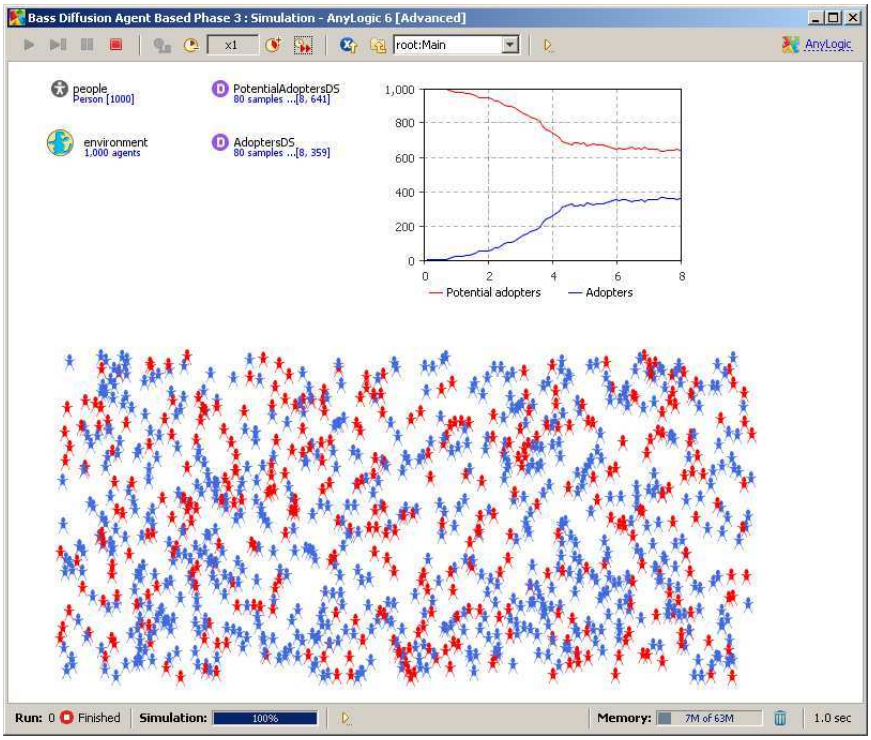
Il est bien adapté à la modélisation de systèmes complexes composés de milliers d'agents agissant en parallèle. Les « modélisateurs » peuvent donner des instructions à un grand nombre d'agents opérant indépendamment les uns des autres, ce qui permet d'explorer les liens entre les comportements des individus à leur niveau et les schémas généraux (comportements de groupe ou de masse) qui émergent des interactions entre de nombreux individus.



c) AnyLogic

AnyLogic permet la modélisation et la simulation de quasiment tous les phénomènes réels grâce à une diversité exceptionnelle [37]. Les modèles AnyLogic peuvent être basés sur n'importe quel paradigme de simulation : événement discret, dynamique de systèmes, et système multi-agents.

Les modeleurs doivent s'intéresser à des approches combinées en vue d'obtenir une vision plus profonde des processus interdépendants complexes et de nature très différente. AnyLogic permet au modélisateur de combiner ces approches de simulation dans le même modèle. Il n'y a aucune hiérarchie fixe. Cette approche de langage combinée est directement applicable à une large variété de problèmes complexes qui peuvent être modélisés via chaque approche indépendamment ou bien avec leurs combinaisons [37].



Il existe d'ores et déjà un grand nombre de plateformes de modélisation et simulation basées sur DEVS, qui semblent (tout du moins celles étudiées) très complètes. La possibilité de vérifier plusieurs modèles est souvent présente, mais l'aspect multi-échelles est moins fidèle.

Conclusion

Cette étude bibliographique nous a permis de constater que le formalisme DEVS de base se révèle très utile pour formaliser des systèmes de toutes sortes grâce à l'apport de cohérence entre les modèles. Cependant, il reste très générique et n'est donc pas adéquat pour la spécification de certains systèmes complexes. C'est pourquoi des extensions variées de ce formalisme ont vu le jour, permettant la modélisation et la simulation de modèles complexes dans des cas précis, ou au contraire s'appliquant à un grand nombre d'applications. Plusieurs extensions de DEVS s'avèrent utiles pour notre

problématique, néanmoins, nous avons pu constater qu'il n'existe pas une extension unique de DEVS qui nous conviendrait pour un laboratoire virtuel de la pompe biologique.

Toujours en utilisant le formalisme DEVS, des outils de modélisation et de simulation ont été développés avec pour objectif l'analyse de systèmes complexes. Ces plateformes, et surtout les laboratoires virtuels, fournissent la possibilité de vérifier plusieurs modèles, comme les modèles dynamiques ou basés agents.

L'objectif du stage va être la conception d'un laboratoire virtuel pour l'étude de la pompe biologique dans le domaine mésopélagique. La modélisation des événements qui régissent ce phénomène va donc être à la base du travail demandé, puis la validation de ces modèles (ou non). Nous aurons besoin d'un formalisme adapté de DEVS et qui répondra à la problématique de l'individu au sein d'une population (niveau micro et macro). La question de l'outil que nous allons utiliser reste ouverte : une plateforme qui existe déjà (VLE, AnyLogic, …), ou bien nous devrons développer notre propre outil générique qui répondrait à nos exigences. Nous pourrons comparer les performances de ces plateformes par la simulation d'un modèle simple de la pompe biologique.

Nous pourrions également nous intéresser aux systèmes multi-interactions (SMI) pour la modélisation de la pompe biologique. En effet, la nature biologique des systèmes modélisés dans un environnement *in virtuo* a conduit à explorer la notion de réification des interactions entre les composants de systèmes biologiques complexes. On est ainsi passé d'une approche de la modélisation initialement centrée individu vers une approche centrée interaction. Les SMA sont adaptés à l'approche individu-centrée, mais l'approche SMI permet de modéliser et simuler certains couplages de phénomènes de façon plus adaptée. Enfin, les SMI permettent à l'utilisateur d'avoir accès à la sémantique du modèle au cours de l'exécution, ce qui en facilite l'observation, la modification et ainsi l'expérience [5]. Cependant, nous n'avons pas trouvé d'extension de DEVS adaptée aux SMI, mais le CERV possède ses propres outils détachés du formalisme DEVS qui pourront s'avérer utiles.

Références

[1] Banas, N. S., Wang, D.-P., & Yen, J. (2003). *Experimental validation of an individual-based model for zooplankton swarming.*

[2] Bergez, J.-E., Raynal, H., & Garcia, F. (2012). *RECORD: an open platform to build, evaluate and simulate integrated models of farming and agroecosystems.* Toulouse, France.

[3] Bratman, M. E. (1999). *Intention, Plans, and Practical Reason.*

[4] Chow, A. C.-H. (1996). *Parallel-DEVS: A parallel, hierarchical, modular modeling formalism and its distributed simulator.* Austin, TX.

[5] Desmeulles, G., Bonneaud, S., Redou, P., Rodin, V., & Tisseau, J. (2009). In virtuo Experiments Based on the Multi-Interaction System Framework: the RéISCOP Meta-Model. *CMES Volume 47 Numéro 3*, 299-329.

[6] Duarte, M., Butz, B. P., Miller, S. M., & Mahalingam, A. (2008). An Intelligent Universal Virtual Laboratory (UVL). *TRANSACTIONS ON EDUCATION volume 51 numéro 1*, 2-9.

[7] Ferber, J., Michel, F., & Baez, J. (2004). AGRE: Integrating Environments with Organizations. *E4MAS*, (pp. 48-56).

[8] Gambiasi, N., Escude, B., & Ghosh, S. (2001). GDEVS: A Generalized Discrete Event Specification for Accurate Modeling of Dynamic Systems. *International Symposium on Autonomous Decentralized Systems*, (pp. 464-469). Washington, DC.

[9] Gao, J., Li, Y., Wang, Y., & Chen, G. (2012). Micro-Macro Modeling for Systems Biology with MR-DEVS. *Applied Mechanics and Materials, 220-223*, 2975-2982.

[10] Goldstein, R., & Wainer, G. (2009). DEVS-Based design of spatial simulations of biological systems. *Winter Simulation Conference*, (pp. 743-754). Austin, TX.
[11] Grimm, V. (1999). Ten years of individual-based modelling in ecology: what have we learned and what could we learn in the future? *Ecological Modelling 115*, 129-148.
[12] Guo, G., Chen, B., Qiu, X. G., & Li, Z. (2012). Parallel simulation of large-scale artificial society on CPU/GPU mixed architecture. *Workshop on Principles of Advanced and Distributed Simulation*, (pp. 174-177).
[13] Hall, C., & Day, J. W. (1977). *Ecosystem Modeling in Theory and Practice.*
[14] Herga, N. R., & Dinevski, D. (2012). Using a Virtual Laboratory to Better Understand Chemistry - an Experimental Study on Acquiring Knowledge. *International Conference on Information Technology Interfaces*, (pp. 237-242). Cavtat, Croatia.
[15] Hong, J. S., Song, H.-S., & Kim, T. G. (1997). A Real-Time Discrete Event System Specification Formalismfor Seamless Real-Time Software Development. *Discrete Event Dynamic Systems Volume 7 Issue 4*, 355-375.
[16] Hwang, M.-H. (2005). Generating Finite-State Global Behavior of Reconfigurable Automation Systems: DEVS Approach. *International Conference on Automation Science and Engineering*, (pp. 254-260). Edmonton, Canada.
[17] Hwang, M.-H., & Choi, B.-K. (2001). GK-DEVS: Geometric and Kinematic DEVS Formalism for Simulation Modeling of 3-Dimensional Multi-Component Systems. *TRANSACTIONS Volume 18 Number 3*, 159-173.
[18] Hwang, M.-H., Cho, S. K., Zeigler, B. P., & Lin, F. (2007). *Processing Time Bounds of Schedule-Preserving DEVS.* Tucson, AZ.
[19] Indraprastha, A. (2011). *Computational Model of Social Interaction in Multi-agent Simulation based on Personality Traits.* Bandung, Indonesia.
[20] Jammalamadaka, R., & Zeigler, B. P. (2007). A Generic Pattern for Modifying Traditional PDE Solvers to Exploit Heterogeneity in Asynchronous Behavior. *International Workshop on Principles of Advanced and Distributed Simulation*, (pp. 45-52).
[21] Kim, T. G. (1995). *DEVS Research at KAIST CORE LAB : From Theory To Practic.* Taejon, Korea.
[22] Li, Y., Li, B. H., Hu, X., & Chai, X. (2011). Formalization of Multi-Resolution Modeling Based on Dynamic Structure DEVS. *International Conference on Information Science and Technology*, (pp. 855-864). Jiangsu, China.
[23] Minar, N., Burkhart, R., Langton, V., & Askenazi, M. (1996). *The Swarm Simulation System: A Toolkit for Building Multi-agent Simulations.*
[24] Muzy, A., & Zeigler, B. P. (2005). *Discrete event simulation of large-scale spatial continuous systems.*
[25] Picault, S., Mathieu, P., & Kubera, Y. (2010). *PADAWAN, un modèle multi-échelles pour la simulation orientée interactions.* Lille, France.
[26] Ragueneau, O. (2010). *Complex Ecosystems: a LAgrangian, VIrtual approach of the ocean carbon biological pump.* Plouzané, France.
[27] Ramat, E., & Preux, P. (2003). "Virtual laboratory environment" (VLE): a software environment oriented agent and object for modeling and simulation of complex systems. *Simulation Modelling Practice and Theory 11*, 45-55.
[28] Ramat, E., Preux, P., Seuront, L., & Lagadeuc, Y. (1998). *Modélisation multi-agents de systèmes naturels - Réflexions générales et application en biologie marine.*
[29] Stratulat, T., Ferber, J., & Tranier, J. (2009). MASQ: towards an integral approach to interaction. *AAMAS*, (pp. 813-820).
[30] Tisseau, J. (2005). *Le virtuoscope - Propositions pour les pôles de compétitivité.* Plouzané, France.
[31] Uhrmacher, A., & Himmelspach, J. (2006). Introducing variable ports and multi-couplings for cell biological modeling in DEVS. *Winter Simulation Conference*, (pp. 832-840). Monterey, CA.
[32] Uhrmacher, A., Ewald, R., John, M., Maus, C., Jeschke, M., & Biermann, S. (2007). Combining micro and macro-modeling in DEVS for computational biology. *Winter Simulation Conference*, (pp. 871-880). Washington, DC.
[33] Vangheluwe, H. (2001). *The Discrete EVent System specification (DEVS) formalism.*
[34] Wainer, G. (2009). *Discrete-Event Modeling and Simulation: A Practitioner's Approach.* Ottawa, Canada.
[35] Wainer, G., & Giambiasi, N. (2001). *Timed Cell-DEVS: modelling and simulation of cell spaces.*
[36] Wilensky, U. (2012). *NetLogo User Manual.*
[37] XJ Technologies. (2005). *AnyLogic Enterprise Library Reference Guide.*
[38] Zeigler, B. P. (1976). *Theory of Modelling and Simulation.* New York, NY.
[39] Zeigler, B. P., & Chi, S. (1991). Symbolic discrete event system specification. *AIHAS.*
[40] Zhang, B., Chan, W. K., & Ukkusuri, S. V. (2011). Agent-Based Discrete-Event Hybrid Space Modeling Approach for Transportation Evacuation Simulation. *Winter Simulation Conference*, (pp. 199-209).