SYSTÈMES MULTI AGENTS
Plate-forme de modélisation MOBIDYC

Guide de l'utilisateur

Version 0.1

Auteurs

Vincent GINOT
Hervé RICHARD

INRA

Unité de Biométrie

Département de Mathématiques et Informatique Appliquées

Centre de Recherche d'AVIGNON

Septembre 2005

Table des matières

Table des figures

Liste des tableaux

Première partie

Introduction

Chapitre 1

Préambule

Dans le domaine de la modélisation d'entités complexes en interactions deux approches sont possibles [1] :

- Développer un système centralisé dans lequel il y a un processus de contrôle qui agit comme pilote des différents évènements exécutés séquentiellements : c'est le principe des *Systèmes experts* .
- Construire des systèmes basés sur l'interaction entre entités relativement autonomes et indépendantes (les *agents*). Ces agents interagissent entre-eux et de ces interactions émergent des structures organisées qui en retour contraignent et modifient les comportements des agents.

Le projet Mobidyc (Modélisation Basée sur les Individus pour la DYnamique des Communautés) se situe dans la seconde catégorie appelée communément *Système Multi Agents* . L'objectif de Mobidyc est de créer, manipuler et modifier des modèles dédiés à la dynamique des peuplements animaux. Il a été initié en 1996 à la suite d'une réflexion menée par un groupe de travail réunissant des chercheurs de l'INRA [1], l'IRD [2], le CIRAD [3] et le LIP6 [4]. Son développement est conduit par Vincent Ginot [5] [2]. Ce projet comporte deux volets :

- Développer un logiciel d'aide à la création et à l'utilisation de modèles individus-centrés qui soit accessible au biologiste. Actuellement une version stable de Mobidyc est en phase de diffusion (version 2.2), parallèlement le projet suit sa phase de développement pour l'ajout de nouvelles fonctionnalités.
- Participer à la réflexion méthodologique sur l'usage de ce nouveau type de modélisation. Son objectif est d'aider le modélisateur à explorer le

1. Institut National de la Recherche Agronomique
2. Institut de Recherche pour le Développement
3. Centre de coopération Internationale en Recherche Agronomique pour le Développement
4. Laboratoire d'Informatique de Paris 6
5. INRA, Biométrie Avignon

fonctionnement de son modèle : analyses de stabilité, analyses de sensibilité, corrélations entre paramètres, sur-paramétrisation, identification...

Ce projet informatique vise à promouvoir l'usage des Modèles Individus-Centrés (MIC). Le choix d'une approche multi-agent facilite le rapprochement avec le point de vue du biologiste car il permet une prise en compte des individus ainsi qu'une visualisation spatiale. Par ailleurs ce projet sert également de support aux travaux de recherches concernant la modélisation de type Individus-Centrés que mène son auteur depuis plusieurs années.

1.1 À propos de ce guide

L'objectif principal de ce guide est de décrire le mode d'utilisation de la plate-forme Mobidyc afin d'offrir aux utilisateurs un support qui complète le didacticiel déjà disponible. Cependant il nous a semblé nécessaire de présenter dans une première partie les principes fondamentaux de la modélisation de systèmes multi agents ainsi que les concepts implémentés dans notre plate-forme. La seconde partie constitue le guide de l'utilisateur proprement dit. L'ensemble des interfaces et des menus y sont décris et l'utilisateur peut directement rechercher la description d'une des fonctionnalités de Mobidyc.

1.2 Droit d'utilisation

L'ensemble du projet Mobidyc est couvert par la licence Gnu General Public License (GPL) (voir en annexe 1 la traduction française)

Nous vous remercions de citer les références complètes du projets ainsi que ses auteurs, pour toute publication ou étude réalisée avec Mobidyc.

1.3 téléchargements

Mobidyc est développé en Smalltalk avec l'atelier de développement **VisualWorksNC** de **Cincom**[6]. Visualworks est décliné en deux versions selon son usage. La version gratuite utilisée pour le développement de Mobidyc est réservée à l'enseignement et la recherche dans un but non-commercial. Pour plus d'information voir le site de Cincom.

À la manière de Java, smalltalk utilise une machine virtuelle qui rend le code multi-système. Pour faire tourner Mobidyc, il suffit de décharger l'archive disponible sur le site du centre de Recherche de l'INRA en Avignon *(www.avignon.inra.fr/mobidyc)* et de l'installer sur un poste de travail. Il est possible de disposer de l'ensemble des sources du projet (en faisant la

6. voir http://smalltalk.cincom.com/index.ssp

demande auprès des responsables)[7], vous pouvez faire toutes les modifications que vous souhaitez à ces sources mais nous vous demandons de citer systématiquement les références du projet Mobidyc (citées en annexes). De plus nous vous serions reconnaissant de nous tenir au courant de vos travaux car nous sommes très intéressés par la confrontation de vos différentes expériences et le feedback est une source précieuse pour l'amélioration du logiciel.

1.4 Historique

1.5 Auteurs et contributeurs

Par ordre Alphabétique :

- **Cyril Blaise :** Génération de tâches (primitives et codage). Gestionnaire des variables.
- **Gerhard Buck-Sorlin :** Traduction des interfaces en allemand, traduction anglaise du tutoriel.
- **Stefan Bornhofen :** Interfaces utilisateur, primitive ModifierAttribut (formules mathématiques).
- **David Cyrus :** Soutien sur ATOLL, optimisation calculs.
- **André Doherty :** Primitives de visualisation des résultats.
- **Vincent Ginot :** Conception / coordination du projet, architecture, tâches, primitives, batch...
- **David Houssin :** Langage de description ATOLL, interpréteur mathématique.
- **Christophe LePage :** Architecture, Espace, Observatoire.
- **Nicolas Kim Nguyen-van :** module de chargement des modèles XML, début de prototypage de JMobidyc (fast Mobidyc en Java)
- **Reza Razavi :** Module Myctalk/Dyctalk d'enchaînement des primitives.
- **Hervé Richard :** Schéma XML, Module de sauvegarde XML, outils divers, maintenance, ce guide, ...
- **Sami Souissi :** Interfaces, Support multi-langue, tutoriel.
- **Cédric Vidrequin :** module java Mathédit. Module basé sur JOME développé par **Laurent Dirat** (voir [16])

1.6 Partenariat

Mobidyc a été initié dans le cadre du groupe de travail *dynamique des peuplements et systèmes multi-agents* associant l'INRA de Thonon (dépar-

7. mailto:mobidyc@avignon.inra.fr

tement hydrobiologie), l'IRD (HEA et LIA), leCIRAD -Tera/Ere et l' Université Paris-VI (LIP6).

- Financement (1996-1999): programme national CNRS *Environnement Vie et Sociétés* dans le cadre de l'action thématique *biodiversitas* du GIP HydrOsystèmes.
- Support de développement actuel : INRA, département de Mathématiques et Informatique appliquées (ex Biométrie et Intelligence Artificielle), unité de statistiques spatiales d'Avignon.
- Collaboration (2005-2007) avec le *Software Engineering Competence Center* du Departement *of Applied Computer Science of the Luxembourg University of Applied Sciences* [8]

8. http://se2c.uni.lu/

Chapitre 2

La modélisation à l'aide de systèmes Multi Agent

La modélisation individu-centrée est de plus en plus employée en écologie et plus particulièrement en dynamique des populations. Elle repose de plus en plus sur le concept informatique de systèmes multi-agents (SMA) qui apporte des ouvertures indéniables en simulation et dont on trouvera un descriptif détaillé dans [5] fait de plus en plus appel par exemple. Pour autant, la mise en œuvre d'un modèle individu-centré et plus encore d'un système multi-agents est délicate et demande un investissement important [6].Force est de reconnaître avec [7] que l'on se trouve soit en présence de de simulateurs trop spécifiques pour répondre à de nouvelles questions, soit en présence de plate-formes effectivement plus génériques mais reposant sur un langage de programmation bien difficile d'accès au non-informaticien.

2.1 Intérêts des SMA

Les systèmes multi-agents (SMA) permettent :

- De prendre en compte la diversité individuelle des êtres vivants et la variabilité spatiale de leur habitat. L'agent peut représenter un individu unique (par exemple un poisson), un groupe (un banc de poissons), ou une population (une densité de plancton), qui évolue si nécessaire dans un espace discret formé d'agents cellules.
- Une grande souplesse dans la définition de la complexité de son modèle. Chaque agent étant autonome, on peut en ajouter, en retrancher, les modifier, sans être obligé de toucher au reste du modèle. Complexifier ou simplifier son modèle devient extrêmement facile et rapide. En comparaison, les modèles plus classiques apparaissent rigides et difficile à construire dès qu'ils sont complexes.

– De rapprocher les points de vue de l'informaticien, du mathématicien et du chercheur autour des mêmes entités, les objets avec leur état et leur comportement. Pour la première fois peut-être dans l'histoire de la modélisation, la structure informatique correspond à la structure biologique que l'on se propose d'étudier. Et il n'est pas anodin de parler le même langage pour mieux se comprendre.

2.2 La solution Mobidyc

2.2.1 Bref résumé de Mobidyc

Mobidyc est destiné à aider un utilisateur non informaticien à créer, utiliser, et faire évoluer ses propres modèles individu-centrés. Basé sur le concept agent, il est bien adapté à la modélisation des systèmes pour lesquels des aspects individuels ou spatiaux sont à prendre en compte. Partant de l'agent de base très simple fournit par la plate-forme, un utilisateur créé ses propres agents, agents de la communauté biologique, agents cellules formant l'espace, ou agents *non-situés* pouvant être chargés de tâches d'intérêt général, en ajoutant de nouveaux attributs pour compléter leur état et des tâches pour fixer leur comportement. Pour ce faire, il assemblera des composants prédéfinis et paramétrables du simulateur qui constituent des sortes d'actions élémentaires appelées *primitives*. Des tâches relativement complexes peuvent ainsi être générées, mais un outil d'aide à la programmation standard est également à sa disposition. Mobidyc gère les variables quantitatives ainsi que leurs unités, les modes de synchronisation séquentiel et (pseudo)parallèle entre agents, ainsi que l'importation de divers fichiers ASCII pour fixer des scénarios prédéfinis sur certains paramètres et charger automatiquement un peuplement initial ou les cartes de valeurs d'une grille environnementale. Un mécanisme de dépendance, permettant de lier à tout instant la valeur d'un paramètre à un autre paramètre via une fonction mathématique analytique ou tabulé, apporte une très grande souplesse dans la constitution des tâches. La sauvegarde des résultats (ainsi que la gestion pseudo-parallèle) est fondée sur le principe que chaque agent est doublé par son *historique* chargé de mémoriser son état (la valeur de ses attributs) à chaque pas de temps. Il est ainsi virtuellement possible de rejouer une simulation et la visualisation et l'exportation des résultats vers un tableur s'en trouve facilitée. Une procédure, particulièrement utile pour les analyses de comportement du modèle, permet d'enchaîner automatiquement des expériences simulatoires (batch). Diverses interfaces permettent de filtrer les résultats à sauvegarder. Si un espace est utilisé, à deux dimensions et selon une grille régulière ou constitué de cellules de formes quelconques, un *observatoire* permet de définir ce que l'on souhaite visualiser durant la simulation.

2.2.2 Les principes architecturaux de Mobidyc

Nous reprenons les grands principes architecturaux de Mobidyc, mais le lecteur pourra se référer à l'article de Ginot et al (2002) [2] pour plus d'information.

Des modèles tout agents

Mobidyc part du constat que dans la plupart des cas, une tâche quelle qu'elle soit suit toujours à peu près la même séquence : il faut localiser l'information, la trier, la traiter, puis mettre à jour le système. Cette séquence se prête bien à un enchaînement de sous actions élémentaires appelées *primitives* (voir la figure 2.1), à condition que *l'information*, qui est un terme très vague, soit correctement structurée et localisée. Une manière radicale de structurer cette information est de considérer que toute l'information contenue dans un modèle doit être portée par des agents. Ce faisant, localiser et traiter de l'information consiste uniquement à localiser et traiter des agents ce qui simplifie énormément le travail des primitives et permet de réduire considérablement leur nombre.

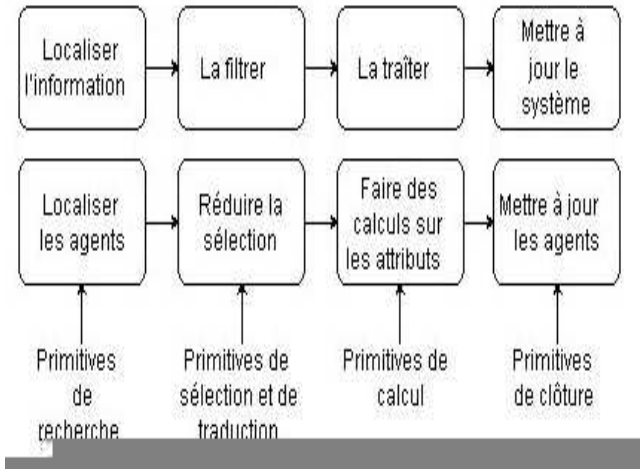

FIG. 2.1: Décomposer une tâche en primitives pour des modèles *tout agents*

Les modèles *tout agents* ont cependant l'inconvénient de forcer à *agentifier* totalement le modèle, ce qui n'est pas toujours très facile. A titre d'exemple, Mobidyc ne sait pas agentifier un espace continu. L'espace est nécessairement vu comme une juxtaposition d'agents de l'espace (agent cellules).

Trois types d'agents

Trois types d'agents seulement sont manipulés par Mobidyc :

- **Les animats :** Terme introduit par Wilson en 1987 et adoptée par l'ISAB, Intern. Soc. for Adaptative Behavior. Ce sont les individus au sens propre, qui peuvent se déplacer, se reproduire ou mourir, selon le comportement que l'utilisateur leur aura donné.
- **Les cellules :** Ce sont les agents de l'espace. L'espace peut donc avoir un état et un comportement, et Mobidyc permet donc de définir facilement des automates cellulaires. Par défaut tout modèle Mobidyc contient au moins une cellule dans laquelle sont placés les agents.
- **Les agents non situés :** Il est fréquent d'avoir à exécuter des tâches d'intérêt général, comme la mise à jour de scénarios climatiques ou le calcul de résumés d'information à destination des agents eux-mêmes ou pour l'utilisateur qui observe les simulations. C'est le rôle des agents non situés. Ils ne sont pas localisés dans l'espace comme les animats, mais sont toujours accessibles par leur nom. Un agent non situé fournit par défaut est l'agent Simulateur, qui porte les informations sur la simulation (pas de temps,, temps écoulé, durée totale) et les rend accessibles à tous les autres agents ainsi qu'à l'utilisateur.

Moules d'agents et agents réels

En programmation standard, le programmeur écrit les classes de ses agents, puis il sera créé autant d'instances de ces classes qu'il faudra créer d'agent réels pendant la simulation. Ici, le programmeur n'écrit pas de classes, mais remplit ce que nous avons appelé des *moules d'agents*, et plus exactement, il complète deux dictionnaires, le dictionnaire d'attributs qui formera l'état de l'agent, et le dictionnaire de tâches qui formera son comportement. Pour passer de ces moules aux agents réels de la simulation, nous avons imaginé un mécanisme qui s'est révélé fort proche des Modèles Objets Dynamiques (AOM) de Riehle et al. (2000) [17].

Deux classes standards sont utilisées, la classe MouleAgent et la classe Agent. On crée une instance de la première à chaque fois que l'utilisateur veut créer un nouvel agent, par exemple un poisson. C'est cette instance que l'utilisateur va remplir en utilisant une bibliothèque d'attributs et une bibliothèques de tâches comme nous le verrons plus loin en détaillant nos exemples (dont bien sûr les tâches qu'il va assembler au moyen des primitives). Une fois le moule complété, les agents réels seront instanciés à partir de la classe Agent. Lors de l'instanciation, l'agent réel ira chercher son état dans son moule via une copie du dictionnaire d'état. Ce faisant, chaque agent est assuré d'avoir son propre état. Pour le comportement par contre, il est juste établi un pointeur vers le comportement du moule : ce faisant tous les agents issus du même moule auront bien le même comportement. Mais il

faut que les tâches qui composent ce comportement puissent accéder à l'état de l'agent qui les possède, ainsi d'ailleurs qu'à l'état des agents qui sont éventuellement impliqués dans ces tâches. Cette fonction est assurée par des liens dynamiques entre les tâches et les états. Nous verrons en section 3.4 comment instaurer ces liens, que nous appelons également dépendances.

Trois mondes et quatre structures pour les agents

Le système manipule donc des moules et des agents réels. Mais il lui faudra aussi manipuler des structures de mémorisation des simulations. Plutôt qu'utiliser un classique système de *ondes* que l'on place sur les entités à mémoriser, nous avons préféré systématiquement doubler chaque agent par son *historique*, i.e. une structure chargée de mémoriser tous les états de l'agent au cours de la simulation. Nous avons même imaginé des primitives spécifiques qui donnent accès à ces historiques : un agent peut donc avoir de la mémoire ! Par ailleurs, pour pouvoir gérer le mode parallèle des agents, il est nécessaire de garder une *image* du monde des agents à la fin du pas de temps précédent, et ceci indépendamment des options de mémorisation. Il y a donc quatre structures pour les agents, les moules, les agents réels, les historiques et les images. Comme ces quatre structures sont très proches, au moins pour leur composante *état*, elle héritent toutes de la classe ProtoAgent. La figure ci-dessus illustre la hiérarchie des classes d'agents de Mobidyc. La classe ProtoAgent gère donc essentiellement l'accès au dictionnaire d'attribut. Elle se

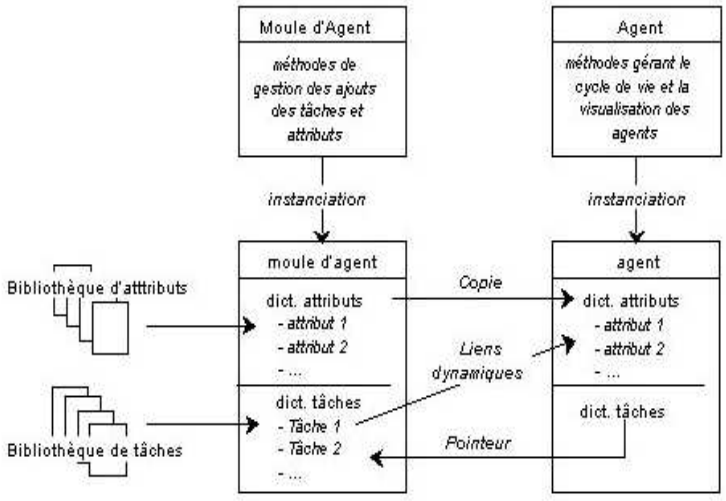

FIG. 2.2: Moules et agents, un concept proche des AOM de Riehle et al., 2000

scinde en 2 sous-classes, la classe Imago qui se voit confier le dictionnaire de tâches, et la classe Image qui ne possède pas de comportement mais assure les liens vers les agents réels (une Image doit pouvoir retrouver son agent). La classe Imago se scinde elle-même en deux sous-classes, les classes Agent et Moule dont nous avons déjà parlé. La classe Image est une classe réelle qui se sous-classe d'une part en ImageCellule qui particularise la classe Image au cas de la cellule, et d'autre part en classe réelle Historique (Memory) qui gère la mémorisation des agents dans le temps.

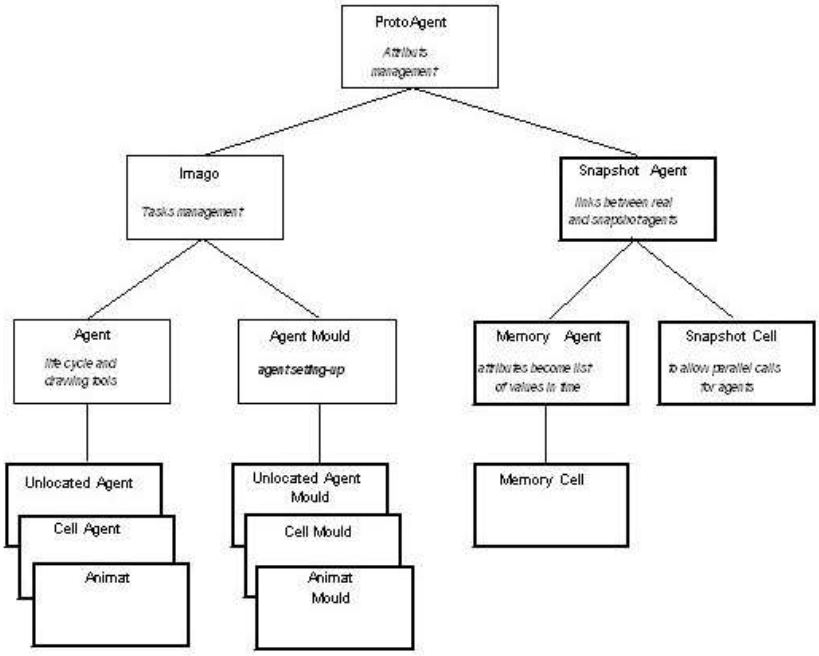

FIG. 2.3: Hiérarchie des classes d'agents de Mobidyc

Les instances de ces quatre structures d'agents sont stockées dans quatre structures, respectivement le *GestionnaireDesEntites* (moules), *LeMonde* (agents réels), *HistoireDuMonde* (agents historiques) et *ImageDuMonde* (agents images). Les trois dernière sont quasiment identiques et héritent d'une classe commune, *StructureDuMonde*. Cette identité de structure, tant sur les mondes que sur les agents, font que les primitives, qui ne s'intéressent qu'aux états des agents, peuvent dans la plupart des cas aussi bien travailler sur une image ou un historique que sur un agent réel. Bien plus, elle peuvent même ignorer dans certains cas dans quel monde elle travaille ! Cette architecture simplifie donc notablement la conception et l'utilisation des primitives, et permet de réduire leur nombre. Nous verrons qu'elle permet en outre de gérer facilement la synchronisation.

Gestion du temps et synchronisation

Mobidyc utilise le mode *horloge* c'est à dire que le temps s'écoule par pas de temps d'amplitudes égales. C'est le mode le plus classique, car facile à programmer et à interpréter. Une alternative beaucoup plus performante mais plus délicate à mettre en œuvre est le mode est le mode *événementiel*, où chaque agent génère des événements sur un calendrier commun. Un événement est un triplet envoyé au gestionnaire du calendrier qui indique l'agent propriétaire, la tâche à effectuer, et la date à laquelle lancer cette tâche. En mode horloge, tous les agents sont censés jouer en même temps (mode d'appel synchrone ou parallèle des agents). Mais ce mode implique souvent de gérer des conflits de ressources (par exemple 2 loups voulant manger une même chèvre). On lui préfère donc souvent le mode séquentiel (ou asynchrone), où l'ordre de passage des agents est tiré aléatoirement : le premier agent à jouer se sert en premier et le second voit le monde tel que le premier l'a laissé. Et ainsi de suite. Mais l'ordre d'appel des agents peut avoir une incidence sur le résultat ?

Mobidyc utilise son architecture particulière pour passer du mode parallèle au mode séquentiel. Le principe en est extrêmement simple : selon le mode choisi, l'ordonnanceur bascule la variable *monde*, soit sur le mode réel (mode séquentiel), soit sur le monde image (mode parallèle). Et toutes les primitives, via l'objet Capteur, lisent dans cette variable *monde*, mais écrivent toujours dans le monde réel.

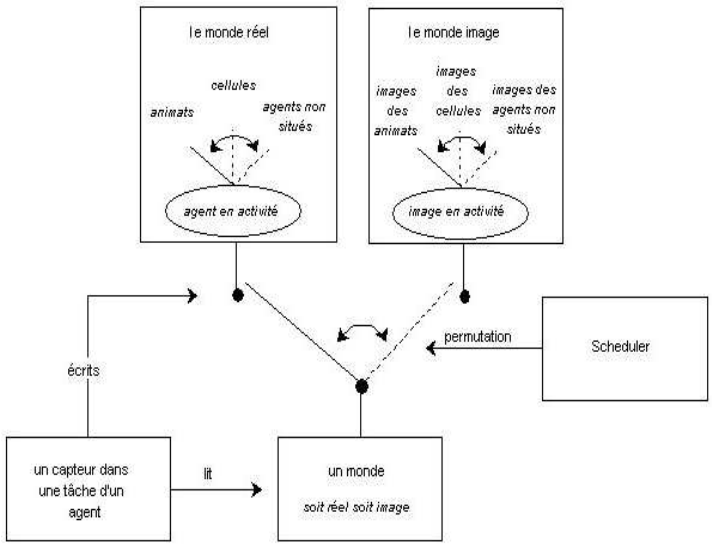

FIG. 2.4: Gestion de la synchronisation à l'aide d'un monde réel et d'un monde image

2.3 Utiliser Mobidyc

La section précédente décrit les grands principes architecturaux qui ont servi de socle à la réalisation de Mobidyc. Ils sont également détaillés dans l'article intitulé **Entre programmation par composants et langage** d'experts publié dans *Technique et Science Informatiques* [12]. Pour résumer on peut dire que :

- toute l'information du système modélisé est porté par des agents réactifs, autrement dit **tout** est agent !
- Ces agents sont caractérisés par un *état* et des *comportements*.
- le comportement des agents est attribué de trois manières possibles :
  - par utilisation de tâches prédéfinies. Par exemple il existe nativement la tâche *vieillir* qui peut être utilisée directement telle qu'elle est, ou alors enrichie en la remaniant via son interface.
  - par assemblage de *tâches élémentaires*. Ces tâches sont prédisponibles dans la plate-forme, elle sont manipulées par l'utilisateur via des interfaces graphiques.
  - par codage. À tout moment l'utilisateur a la possibilité de coder son propre comportement en écrivant du code smalltalk qui sera directement pris en compte dans la plate-forme.

Construire un nouveau modèle

Un modèle sous Mobidyc se fait en principe en six étapes :

- Donner un nom de projet
- Définir son système d'unités
- Définir les cellules de l'espace sont des agents
- Initialiser et placer des agents pour débuter une simulation
- Définir le pas de temps et la durée de simulation
- Lancer et éventuellement poursuivre une simulation.

Ces étapes seront détaillées en même temps que les boîtes de dialogues seront passées en revue.

2.3.1 Outils et interfaces

Le point fort de Mobidyc et de proposer au biologiste des boîtes de dialogues pour manipuler les composants. De plus toute une gamme d'outils est également disponible pour :

- gérer les différents modèles
- contrôler les paramètres des modèles
- configurer les simulations
- visualiser les simulations

– créer des lots de simulations
– analyser les résultats
– mémoriser les simulations
– ...

2.4 Les fichiers nécessaires

La plate-forme Mobidyc nécessite un ensemble précis de fichiers organisés selon le schéma de la copie d'écran de la figure 2.5.

2.4.1 à la plate-forme

image Le répertoire image contient tous les fichiers indispensables au fonctionnement du programme, soit :

- **mobidyc.im :** le fichier image du code Smalltak de Mobidyc
- **mobidyc.cha :** le fichier qui conserve l'historique de l'image
- **grammaire.gra :** *à quoi cela sert-il ?*
- **mobidyc.loc :** Généré au premier lancement de Mobidyc, il contient le flag précisant la langue utilisée.
- **Fichiers spécifiques à MathEdit :**
  - **MathEdit.jar :** Ce jar contient l'éditeur mathématique java.
  - **MathMlTagToFct.xml :** Ce fichier contient une liste prédéfinie de fonctions mathML utilisées par celui-ci.
  - **tagTranslation.xml :** Ce fichier contient une liste de conversion des tags selon la langue utilisée.
-

exemples Ces dossiers (un par langue) contiennent les jeux de fichiers nécessaires aux exemples données.

MB_plugin Ce dossier contient les patchs qui sont éventuellement livrés pour correction de bug.

messages Ce dossier contient toute l'arborescence nécessaire à l'affichage de l'aide Mobidyc et mathscript. Il faut penser à vérifier sa bonne installation si on a des problèmes d'affichage de l'aide.

projets C'est le répertoire par défaut ou sera sauvegardé les nouveaux modèles développés par l'utilisateur.

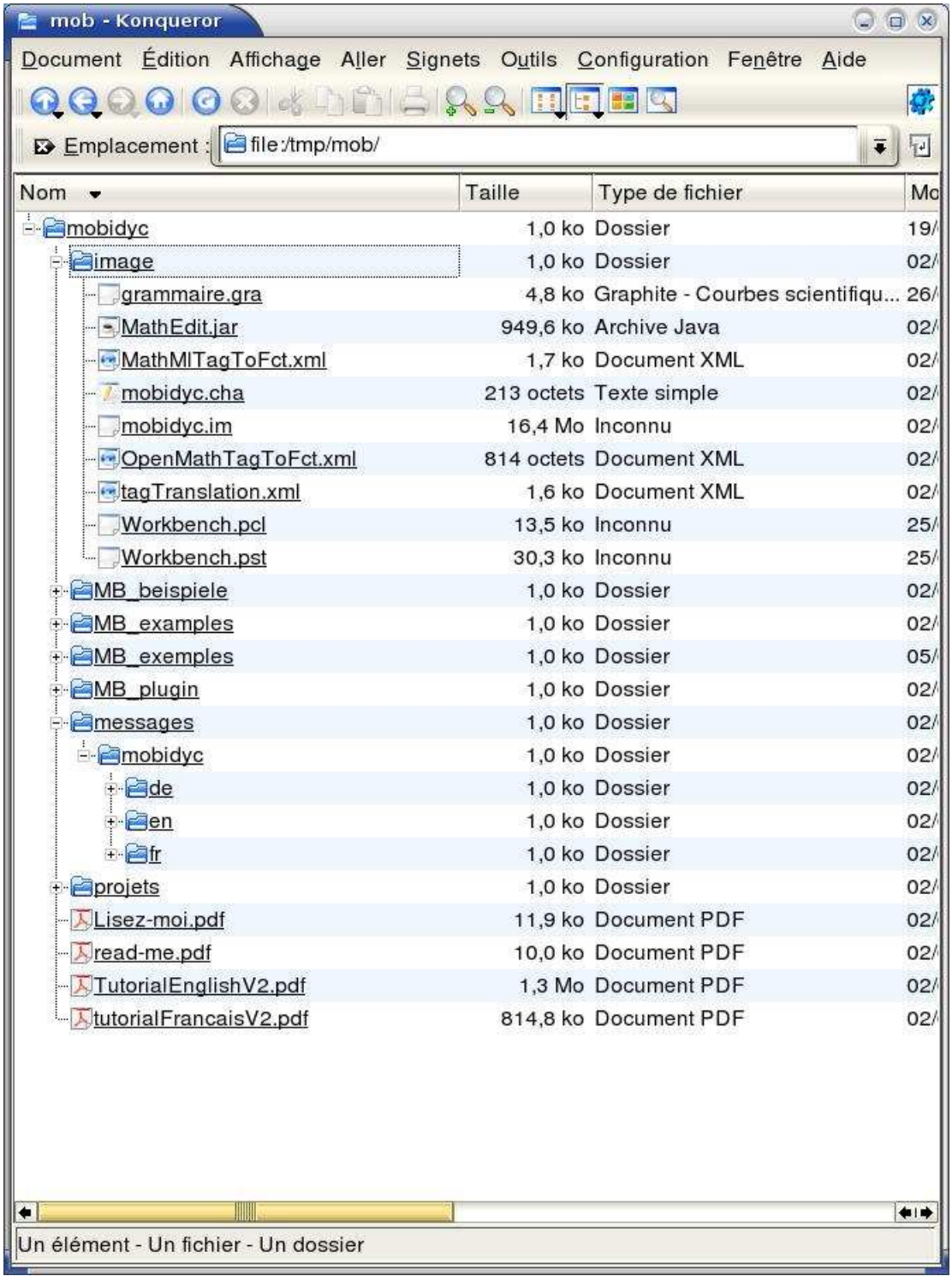

FIG. 2.5: arborescence de la plateforme Mobidyc.

2.4.2 à un projet

Le fichier .mbp, il contient la référence à tous les autres fichiers et tout ce qui est lié à la gestion des simulations. Il sauvegarde en particulier l'ordonnanceur (scheduler), les préférences et les expériences simulatoires.

Le fichier .ppl, fichier des moules. Ce fichier contient l'objet *gestionnaire d'entités* du simulateur, c'est à dire principalement les moules de tous les agents.

Le fichier .esp, fichier de l'espace. Ce fichier contient le descriptif de l'espace (taille de la grille, type de cellules etc.), puis la valeur initiale des attributs de toutes les cellules. Donc une ligne par cellule. On peut l'éditer pour changer *à la main* les caractéristiques de la grille. Par exemple pour changer la taille de l'image, ou la topologie de la grille, Mobidyc ne permettant pas de le faire actuellement.

Le fichier .pdv, fichier des observatoires. Ce fichier contient un *Dictionnaire des points de vue* qui rassemble l'ensemble de point de vues que vous avez pu définir sur les agents et leurs cellules permettant une visualisation sur les cartes 2D.

Le fichier .chg, fichier du chargement initial des animats. Ce fichier ressemble beaucoup au fichier de l'espace. Il contient le descriptif initial des animats, Il sera créé autant d'agents que de ligne de valeurs.

- 1ère ligne, rappel du nom du fichier et du codage ASCII.
- 2ème ligne, mot réservé *agent*, puis le nom complet de l'agent.
- 3ème ligne, nom des attributs. 4ème ligne unité (optionnelle) des attributs.
- Lignes suivantes, valeurs des attributs. On passe à l'agent suivant en reprenant le mot réservé *agent* et le nom de l'agent suivant Et ainsi de suite.

Les fichier .pcl et .pst, fichier de sauvegarde des Tâches. Ces fichiers sont duaux, l'un contient le code binaire (pour gagner du temps au chargement du projet), l'autre le code source au format XML. Ils contiennent la sauvegarde des tâches formules et fonctions générées au cours des différents projets. Ces fichiers, qui sont donc associés à chaque projet, remplacent le fichier TachesUtilisateur.pcl (et .pst) des versions antérieures de Mobidyc. Ce fichier est néanmoins conservé (dans le répertoire image) pour la compatibilité des anciens modèles. Mais dès qu'un ancien modèle est sauvé, les fichiers nomProjet.pcl et nomProjet.pst sont automatiquement créés et complétés.

Le fichier .xml, Ce nouveau fichier est la transcription XML selon notre grammaire des modèles Mobidyc. C'est le seul qui permette la relecture d'un modèle plus ancien sur une plate-forme Mobidyc modifiée moyennant un minimum de correction dans le code XML. La sauvegarde dans ce fichier est systématique lorsqu'on sauvegarde le projet.

Ces 8 fichiers (si tous présents) représentent l'intégralité d'un projet. Attention de bien tous les inclure, en particulier dans le cas d'un transfért du modèle sur une autre machine.

Après cette brève entrée en la matière la suite du document sera consacrée à la description de l'ensemble des fonctionnalités. La démarche choisie consiste à décrire les boîtes de dialogues et les fonctionnalités associées des niveaux les plus élevés vers ceux les plus spécifiques tout en respectant la logique de construction d'un modèle.

Les boîtes de dialogues sont appelées à travers les fonctionnalités des différents des menus. Par ailleurs il existe des menus contextuels (typiquement dans les champs de saisie ou dans les listes). On y accède avec la clé *alt clic* sur mac, *clic milieu* sous Unix/Linux, *clic droit* sous Windows.

2.4.3 Nouveautés

entre la version 2.2 et 2.2

- Possibilité de sauvegarder et de relire les modèles au format XML. Le premier intérêt est de rendre la sauvegarde des modèles plus sûr : on garde une trace écrite (ASCII) du modèle, et on est assuré de pouvoir faire relire ce modèle par les prochaines versions de Mobidyc. A moyen terme, nous envisageons de développer des *résumés* automatiques textuels des modèles, afin que de pouvoir les communiquer plus aisément. A plus long terme, on peut imaginer que ces fichiers XML puissent être relus par des plate-formes d'exécutions optimisées pour leur vitesse de calcul.
- Intégration d'un éditeur de formules mathématiques en remplacement de la primitive *ModifierAttribut*. Cet éditeur pleine page intègre les variables temporaires et les structures de contrôles (conditions et boucles). Il génère automatiquement une tâche Mobidyc qui s'exécute sensiblement plus vite que ModifierAttribut. Cet éditeur est écrit en Java, et utilise un formalisme XML d'échange avec Mobidyc basé sur MathML. Il nécessite Java Runtime Environment.
- Accès et gestion des projets Mobidyc basé sur la boîte de dialogue graphique *FileBrowser* de Cincom VisualWorks. Il est enfin possible naviguer librement dans les dossiers pour rechercher et sauvegarder des modèles.

- Intégration d'un Tableau de Bord pour gérer l'ensemble des paramètres que le modélisateur souhaite faire varier lors de ses simulations. On a ainsi sous les yeux la valeur de tous les paramètres importants, et il est possible de les modifier à tout instant.
- Nouveaux boutons dans la barre d'outils, en particulier *sauvegarde rapide*.
- Optimisation des écritures des fichiers de résultats lors des batchs
- Un nouveau plan d'expérience, le plan complet automatique.

entre les versions 2.0 et 2.1

- Utilise le moteur Smalltalk VisualWorks version 7.2 de Cincom (contre version 7.0).
- Possibilité d'importer des fichiers ASCII de contours de cellules de formes quelconques. Mobidyc calcule automatiquement le voisinage de chaque cellule. Essayez le fichier *polygone.cel* fourni avec les exemples, et qui vous donne le format à utiliser.
- Possibilité de modifier un assemblage de primitives en cours d'utilisation dans un agent, sans perte du paramétrage des primitives. Jusqu'à présent, il fallait créer un nouvel assemblage puis reparamétrer toutes les primitives (menu contextuel *Éditer assemblage*).
- Création d'une nouvelle tâche prédéfinie, la tâche Alias. Comme son nom l'indique, placée dans un agent, elle pointe vers une tâche d'un autre agent. Cela permet à deux types d'agents différents de partager la même tâche. Utilisable par tout type d'agent.
- Possibilité que chaque cellule puisse gérer ses propres scénarios, via la nouvelle tâche *ScénarioDansAttribut* et le nouveau type d'attribut *SerieXY*. Jusqu'à présent un scénario était toujours commun à toutes les cellules. Les attributs *SerieXY* sont également accessibles par une fonction tabulé particulière, la fonction *FonctionTabuléeDansAttribut*.
- Exportation textuelle des résultats concernant les cellules. Donne un tableau par attribut, avec les cellules en lignes et les pas de temps en colonne.
- Meilleure synchronisation entre la visualisation des résultats et l'état de la simulation.
- Utilise le moteur Smalltalk VisualWorks version 7 de Cincom (contre version 5i4).
- Gestion différente de la sauvegarde des tâches utilisateurs. Elles ne se trouvent plus dans la parcelle *TachesUtilisateurs.pcl* (répertoire image), mais sont associées à chaque projet sous le nom nomProjet.pcl (et nomProjet.pst pour les sources). Avantage, quand vous exportez un modèle, vous exportez également ses tâches utilisateurs. Inconvénient, vous ne

pouvez plus utiliser une tâche d'un modèle à l'autre. La parcelle TachesUtilisateurs reste présente pour compatibilité avec la version 1.
- Un item *copier/coller* permet de recopier une tâche complète d'un agent à l'autre.
- Les noms de formules et de fonctions peuvent maintenant utiliser des chiffres.

2.4.4 Bugs Connus

Toutes plate-formes confondues

-
- Éditeur MathScript. Si rien ne se passe lorsque on ajoute une tâche/primitive MathScript aux agents, c'est vraisemblablement parce que la Java runtime environment n'est pas installée sur la machine (MathScript utilise Java). MathScript nécessite java runtime J2SE 1.4.2 ou plus récent, à télécharger sur le site http://java.sun.com/j2se.
- Boîte De Dialogue de chargement initial des agents : les clics directs entre les moules d'agent et leurs valeurs (tableau du milieu et tableau de droite) provoquent une erreur. Pour l'éviter, après avoir saisi une valeur dans la table du milieu, cliquer plus à gauche pour sélectionner toute la ligne. Vous pouvez ensuite cliquer sans danger sur le tableau de droite.
- Utilisation des fonctions l*n()* et *exp()* (éditeur de formules). Mobidyc traduit les unités dans le système d'unité par défaut.
  Attention aux fonctions *log* et *exp* : Si par exemple on calcule $log(taille)$ avec taille en mm, et que l'unité par défaut est le mètre, Mobidyc traduit la taille en mètres avant de passer au log ce qui peut surprendre... Il est possible de changer les unités par défaut via le menu Préférences− >unités.

avec Windows

- Sur certaine machines (e.g. Windows 2000 pro), lors de grosses expériences simulatoires, les simulations peuvent ralentir après quelques milliers de simulations. Un correctif existe, nous consulter.
- Boîte De Dialogue de la tâche Batch, bouton *Pré-Action*. Si cette préaction est un assemblage de primitives, sur certaines machines, il peut être impossible de sortir correctement de son interface. Car elle attend la validation (bouton *valider*) alors que Mobidyc est repassé sur la fenêtre Batch qui ne réagit pas non plus au bouton *valider*. Pour se sortir d'affaire, faire glisser la fenêtre Batch au dessus de la fenêtre de paramétrage de l'assemblage de primitives, et cliquer, dans la fenêtre Batch, au niveau du bouton *valider* de la fenêtre d'en dessous.

En visant bien, cela permet de sortir de la fenêtre de l'assemblage des primitives.

avec Mac

- MathScript ne fonctionne pas sous Mac OS 9.
- Les menus contextuels ne fonctionnent pas dans l'éditeur MathScript. Utiliser la barre de menu.

Deuxième partie

Manuel de l'utilisateur

Chapitre 1

L'interface principale

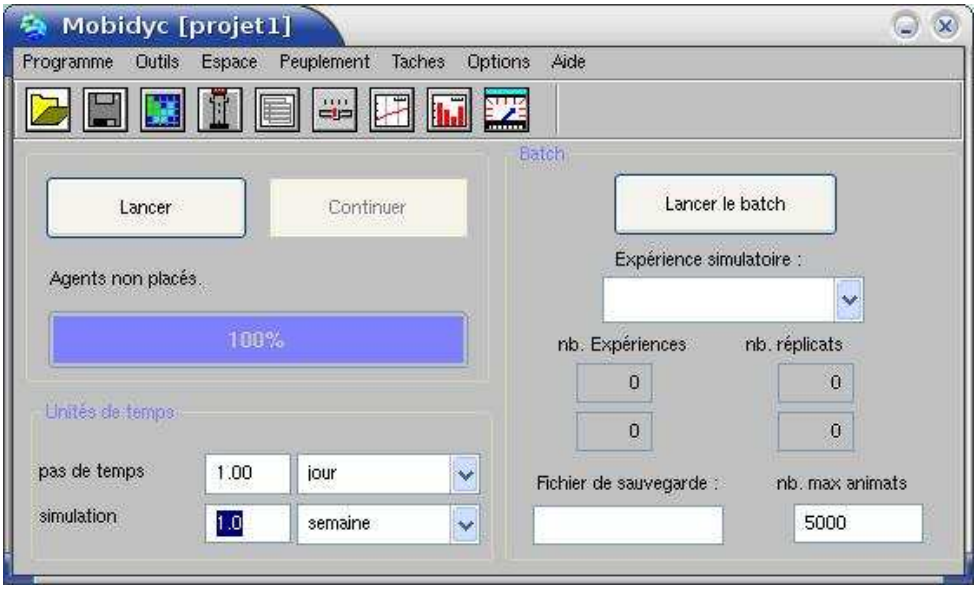

FIG. 1.1: Interface principale de Mobidyc avec le nom du projet rappelé dans le bandeau de la fenêtre (par défaut : *Projet1*).

L'interface principale de Mobidyc (figure 1.1) est une fenêtre divisée en 5 groupes; En allant du haut vers le bas de la fenêtre on trouve :

- une ligne de différents menus qui permettent d'exécuter les commandes de Mobidyc.
- une bande contenant des icônes correspondant aux raccourcis des fonctions les plus utilisées
- un groupe dédié à l'exécution de la *simulation* des modèles
- un groupe dédié à la gestion de lots de simulations ou *Batch*

– un groupe dédié à la gestion du *temps* des simulations

1.1 Ligne des menus de commandes

Toutes les fonctions de Mobidyc peuvent être activées en sélectionnant le bon *item* proposé dans le menu de la ligne de commande de l'interface principale. Celui-ci est divisé en 6 grandes catégories que nous allons passées en revue :

- **Programme** Contient les différentes méthodes nécessaires à la gestion et l'exécution des modèles.
- **Outils** Offre une panoplie d'outils qui servent à la visualisation et à la gestion de l'exécution des modèles.
- **Espace** Regroupe toutes les fonctionnalités pour la gestion et la création des *agents cellules*.
- **Peuplement** gère la création et la modification des *animats* et des *agents non situés*, ainsi que le placement initial des animats avant simulation.
- **Tâches** gère la création et la modification des tâches créées par l'utilisateur.
- **Options** propose d'affiner la configuration du logiciel en fonction des choix e l'utilisateur.
- **Aide** offre l'aide en ligne. Il également noter que (presque) chaque boîte de dialogue appelée possède un bouton d'aide.

1.2 Bandeau des raccourcis

Ce bandeau offre des pictogrammes et une information textuelle lorsque la souris survole la zone correspondante aux principales fonctions de Mobidyc. On a :

- Charger un projet *(voir section 2.3.2)*
- Sauver un projet *(voir section 2.3.2)*
- Visualisation *(voir section 2.4.2)*
- Observatoire *(voir section 2.4.1)*
- Scheduler *(voir section 2.4.3)*
- Magnétoscope *(voir section 2.4.4)*
- Chronique *(voir section 2.4.5)*
- Histogramme *(voir section 2.4.6)*
- Tableau de bord *(voir section 2.4.7)*

1.3 Groupe *simulation*

La zone *Simulation* située à gauche sous le *Bandeau des raccourcis* est composée de 2 boutons, d'un champ d'informations textuelles et d'un *ascenseur* indicateur de progression.

1.3.1 Boutons

Une fois un modèle défini, l'utilisateur lance la simulation à partir du bouton **Lancer** de cette zone (du moins pendant la mise au point). Le bouton **Continuer** est *gris* c'est à dire inhibé.

Lorsque la simulation est lancée, le bouton **Continuer** est devenu actif avec le label **arrêter**.

Une fois la simulation achevée, le bouton de gauche propose de **Poursuivre**. Si on clique sur ce bouton on va lancer une seconde simulation qui se concaténera à la première (on remarque que le nombre d'agents double).

1.3.2 Informations

Au départ la ligne affiche *Simulation prête à lancer.*,
En cours de calcul on lit *Simulation tourne... xxx/yyy (z)* avec :

- xxx : le numéro d'itération de la simulation en cours
- yyy : le nombre total d'itération.
- z : le nombre total d'agents restants.

En fin la ligne *simulation terminée* est affichée.

1.4 Groupe *Batch*

Le batch (ou lot) permet d'enchaîner automatiquement des simulations dans ce qu'il est convenu d'appeler des *séries d'expériences simulatoires*. Ce mode permet de faire varier des paramètres et de sauver les résultats. La configuration des expériences simulatoires se fait à l'aide du menu contextuel qui apparaît lorsque le pointeur de la souris est dans le champs *Expérience Simulatoire* **et** qu'on clique sur le bouton droit. Mobidyc propose plusieurs types d'expériences :

- Exploration paramètre par paramètre
- Plan d'expérience complet (pour ANOVA)
- Plan d'expérience complet automatique
- Plan d'analyse de sensibilité
- Hyper cube latin
- Saisie manuelle du plan.

De plus on peu *ouvrir* une expérience déjà sauvegardée, ou alors en *supprimer* une.

Si des expériences ont déjà été définies, la liste apparaît dans le menu déroulant de droite. Il faut alors sélectionner la série d'expériences souhaitée, puis sélectionner *ouvrir l'expérience* du menu contextuel pour la modifier.

Principe de fonctionnement du batch Du fait que les agents partagent tous le même dictionnaire de tâches (stocké dans le moule des agents) modifier un paramètre dans une tâche provoque immédiatement une modification de tous les agents concernés par la tâche... Ceci permet de *batcher* très facilement !

Corollaire, si ce paramètre est lié par une dépendance (simple) à un attribut d'un agent, la valeur du paramètre que l'on souhaite modifier n'est plus accessible car chaque agent en est propriétaire. On tourne la difficulté en utilisant une dépendance complexe (via le menu contextuel) pour le paramètre de la tâche. Il faut mettre une dépendance de type *paramètre + fonction identité* par exemple.

En procédant ainsi la valeur du paramètre reste interne à la tâche, et représentera par exemple la valeur moyenne du paramètre sur laquelle on pourra batcher. La partie dépendance pointe (ici via la fonction identité) vers l'attribut de l'agent. Notons que ce dernier ne porte plus la valeur du paramètre, mais l'écart à la moyenne.

Les résultats sont sauvegardés dans un fichier dont le nom est passé dans le champs *fichier des sauvegarde*. Seuls les *Batch* sont sauvegardés. Voir le chapitre 4 pour le fonctionnement détaillé des différentes expériences simulatoires.

NB Pour ceux qui utilisent les logiciels de statistiques Splus ou R, nous démarrons un projet de module de dépouillement de ces expériences. Nous contacter.

1.5 Groupe *Gestion du temps*

4 champs permettent de fixer le *pas* et l'*unité* de temps, ainsi que la *durée* de la simulation dans une unité donnée. Les unités de temps sont choisi parmi une liste finie.

1.6 En cas d'erreur

En cas d'erreur logiciel, un message invite à essayer de *continuer* (Proceed, déconseillé), d'*annuler* l'action en cours (Continue), ou de quitter le

programme (Exit). Un fichier d'erreur (error.log, répertoire image) est généré. Il peut aider à comprendre le problème. Détruire ce fichier (car il est en mode *ajout*) avant de relancer Mobidyc.

En cas d'urgence, on peut manuellement interrompre Mobidyc en pressant *Ctrl Y*. Vérifier toujours l'état du modèle. En particulier, si on interrompt une procédure Batch, recharger immédiatement le projet pour restaurer les valeurs par défaut de votre modèle.

Chapitre 2

Les menus déroulants

2.1 Aide

Classiquement ce menu offre des informations sur le logiciel, on y trouve :

- *À propos de Mobidyc* : des informations générales (voir le préambule de ce guide, chapitre 1)
- *Aide* : L'aide proprement dit avec une liste de mots-clés qui focalisent la recherche
- *Nouveautés* : permet de spécifier les nouveaux ajouts
- *Bugs connus*

2.2 Options

Le menu **Options** est celui qui devrait être activé en premier lorsqu'on débute avec Mobidyc car il permet de configurer la plate-forme de l'utilisateur en fonction de son environnement et de ses habitudes de travail.

2.2.1 gestion de la langue

Lors du tout premier lancement de Mobidyc l'utilisateur est forcé de choisir la langue de dialogue de Mobidyc. Il peut à tout moment choisir de basculer dans une autre langue. Dans ce cas Mobidyc se fermera et s'ouvrira de nouveau dans la bonne langue. Il faut noter que les *Projets* (i.e. les Modèles Mobidyc) sont dépendant de la langue, par conséquent il est impossible d'ouvrir un projet en anglais si la langue choisie pour exécuter Mobidyc est le français (et réciproquement). Une manière drastique de forcer Mobidyc à demander à l'utilisateur de choisir sa langue est de détruire le fichier *mobidyc.loc* dans le répertoire contenant l'image Mobidyc.

2.2.2 Préférences

Ce menu permet de fixer des choix concernant :

la visualisation de l'espace

Elle se paramètre de plusieurs manières, ce peut être :

- en fixant le nombre de pixel caractérisant la taille de la fenêtre (400 par défaut). *ATTENTION : cette fonctionnalité ne marche pas !*
- en modifiant la taille des Animats; deux cas sont alors possibles :
  - soit le modèle met en œuvre des fichiers de contours et dans ce cas il peut fixer la taille absolue (en nombre de pixel) des animats
  - soit le modèle manipule des animats directement dans l'espace, alors il est possible de fixer le ratio animat/cellule qui caractérise la grosseur du point de représentation de l'animat dans sa cellule
- en choisissant de masquer ou d' afficher la visualisation des contours des cellules de l'espace (seules les couleurs les distinguerons).

Cliquer sur le bouton *Appliquer* pour valider ces choix.

Batch

Par défaut et par souci d'économie de mémoire le mécanisme de Batch désactive les mécanismes d'historique (il n'a donc plus de magnétoscope ou autres fonctionnalités basées sur ce mécanisme).Certaines primitives nécessitent de conserver ce mécanisme. Il faut donc penser à activer celui-ci si le modèle utilise ces primitives (voir la partie traitant des primitives, chapitre 3, section 3.4).

Unités

Cette boîte de dialogue permet de sélectionner les unités de *temps*, *longueur* et *poids*. Cliquer sur le bouton *Appliquer* pour valider ces choix.

2.2.3 options du développeur

Il est possible de mettre à jour les fichiers d'aide et les dictionnaires en particulier lorsqu'on a développé de nouvelles fonctionnalités (de nouvelles primitives par exemple). Ceci se fait à l'aide de ces différents item.

2.2.4 Patch

Lors de la correction d'un ou plusieurs bug, plutôt que de refaire une distribution complète que chacun doit de nouveau télécharger et installer il paraît plus judicieux d'avoir un mécanisme de chargement et de déchargement du code correctif. C'est ce que ces 2 item proposent.

ATTENTION il faut que le code source de ces ajouts se trouve dans le répertoire ***plugin*** *de l'arborescence Mobidyc.*

2.3 Programme

Le menu programme contient cinq groupes d'item.

2.3.1 Caractéristiques pour un projet

Nouveau projet...

Ce choix permet de spécifier le nommage d'un projet (le projet par défaut chargé au lancement de Mobidyc s'appelle *projet1* (nom que l'on retrouve dans la bannière de la fenêtre).

Unités du projet

Cet item rappelle l'interface du menu *options»préférences* pour spécifier le choix des unités du modèle.

2.3.2 lecture/sauvegarde d'un projet

Charger un projet

Lorsqu'on souhaite charger un projet de modélisation existant, on utilise la boîte de dialogue *projectBrowser* qui manipule l'ensemble des fichiers nécessaires à la description d'un projet.

La boîte de dialogue de la figure 2.1 est héritée de l'interface native *fileBrowser* de *visualWorks* ce qui explique que les textes des menus soient seulement en anglais. Nous ne décrirons que les commandes et champs spécifiques à la gestion de projet.

La boîte de dialogue se présente en trois zones :

- L'arborescence du disque, sur la gauche.
- le contenu du répertoire selon le filtre sélectionné, en haut à droite.
- Si aucun fichier n'est sélectionné dans la fenêtre *Haut à Droite* :
  - Le contenu exhaustif du répertoire
  - les informations spécifiques concernant les accès du répertoire.
- Si un fichier est sélectionné dans la fenêtre *Haut à Droite* :
  - le contenu *brut* du fichier : *index* ***Text***
  - le code hexal décimal du fichier : *index* ***Binary***
  - les informations d'accès du fichier : taille, chemin complet, dernier accès... : *index* ***File Information***

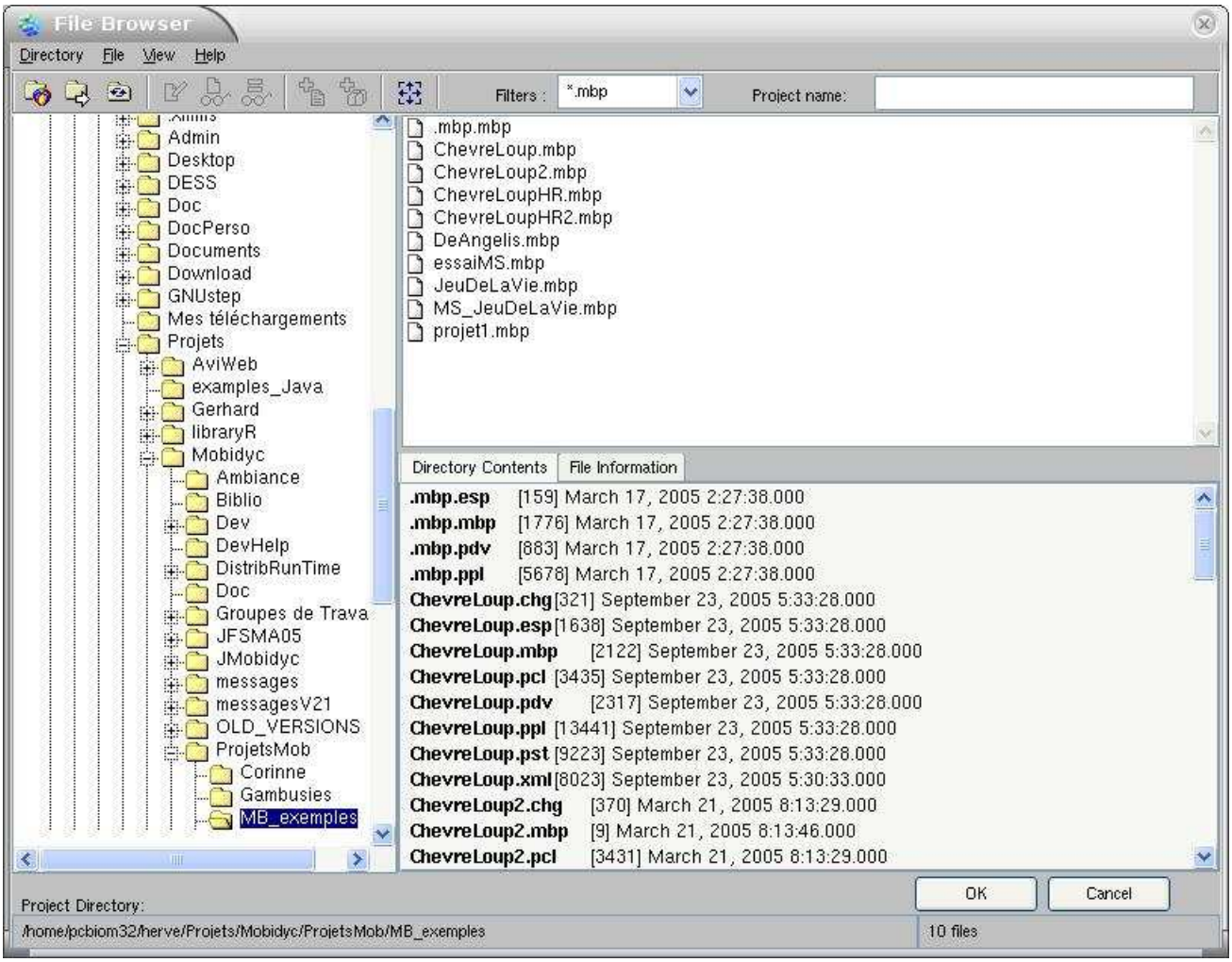

FIG. 2.1: Interface de gestion des fichiers projets.

Nous appellerons *Barre d'icônes* la barre qui se situe immédiatement en dessous de la barre de menu. Cette barre propose 2 champs sur la droite :

- *Fliters* : dans lequel on choisit le type de fichiers que l'on veut sélectionner. Par défaut c'est les fichiers de type **.mbp** qui sont listés.
- *Project name* : affiche le nom du projet de modélisation courant.

Sur le bas de la fenêtre se trouve :

- Sur la droite le chemin complet du répertoire contenant les fichiers du projet en cours
- Sur la gauche les boutons de *validation* ou *d'abandon* de la boîte de dialogue.

dans une session Mobidyc le nom du **répertoire projet** est conservé, ce qui permet de pouvoir directement afficher ce répertoire quand la boîte de dialogue *project Browser* est appelée. Dans le cas ou ce répertoire n'est pas identifié, c'est le répertoire qui contient l'image courante qui est pris comme valeur de répertoire *projet courant*.

Sauver un projet sous. . .

Cette fonction appelle la boîte de dialogue *Project Browser* pour donner explicitement un nom au projet à sauvegarder.

Sauver un projet

Sauvegarde le projet sous son nom en cours (par défaut **projet1**). *Attention aucune confirmation n'est demandée et la sauvegarde écrase les fichiers du même nom existants.*

Il est conseillé de sauvegarder régulièrement son travail afin de pouvoir repartir de fichier les plus récent possibles en cas de problème.

2.3.3 formalisme en XML

Exporter un projet vers XML

cette fonction exécute une sauvegarde explicite du projet dans le formalise XML utilisé pour décrire les modèles Mobidyc. Le résultat est un fichier XML. Notez toutefois que chaque sauvegarde classique du projet entraîne la création du fichier XML.

Compiler un projet en XML

Cette fonction permet l'import d'un projet stocké dans le formalisme XML. Cette fonctionnalité peut être très utile pour reprendre un projet échangé avec un autre utilisateur.

2.3.4 simulation

Lancer la simulation

Cet item du menu permet d'exécuter une simulation une fois qu'un modèle est défini. Un accès rapide à cette commande se fait grâce au bouton **Lancer** du groupe *simulation*.

Lancer sans processus

Par défaut, lorsqu'on lance une simulation la machine virtuelle Smalltalk *fork* (dédouble) le processus et le calcul se fait dans le nouveau processus. Cette méthode à l'avantage d'isoler le calcul dans un nouveau processus que l'on peut stopper avec un bouton, mais elle est un peu plus coûteuse en terme de CPU. Il peut être intéressant de pouvoir optimiser au maximum le temps de calcul en lançant le calcul dans le même processus, c'est ce que permet de faire cet item. Dans ce cas, la seule façon d'interrompre le calcul est de faire un **ctrl-y**. Notons que la gestion des lots de simulations se fait différemment (voir la section 2.3.4).

Retour à t=0

Lorsqu'on lance une simulation est toujours possible de l'arrêter durant le déroulement des calculs (voir le paragraphe sur la description du groupe *Simulation*, section 2.3.4). Pour cela il faut utiliser le bouton *Continuer* du groupe qui se transforme en bouton *Arrêter* durant le calcul d'une simulation. Il est également possible de relancer la simulation à son départ c'est ce que permet la commande *Retour à t=0*.

Mémoriser la simulation

Une fois un ou plusieurs cycles de simulations réalisés, il est possible de mémoriser celles-ci dans un fichier. En lançant cette commande la boîte de dialogue de gestion des projets s'ouvre et propose la création d'un fichier du nom du projet et suffixé par *.sim*.

Charger une simulation

Une simulation mémorisée dans un fichier peut être rechargée grâce à cette commande.

2.3.5 quitter

C'est la seule façon *correcte* de sortir de Mobidyc. Tous les fichiers ouverts sont fermés correctement.

ATTENTION : Il n'y a pas de demande de sauvegarde du projet, toute modification de celui-ci est perdu sauf d'avoir réalisé une sauvegarde explicite avant.

2.4 Outils

2.4.1 Observatoire

La figure (figure 2.2) représente la boîte de dialogue de définition des *points de vue* qui permet de spécifier la façon dont on souhaite visualiser les agents du modèle. Il est possible de visualiser les agents systématiquement (choix *Vue sur l'agent*) ou de ne les visualiser que sous certaines conditions portant sur l'état de l'agent (choix *Vue sur des attributs de l'agent*. Dans ce cas une liste de conditions (avec opérateur) se construit en suivant les menus contextuels du tableau qui apparaît lorsque on choisi la *vue sur des attributs des agents*. Il est par exemple possible de ne visualiser que les agents dont le poids est supérieur à 100 kg et dont la taille est inférieure à 50 cm.

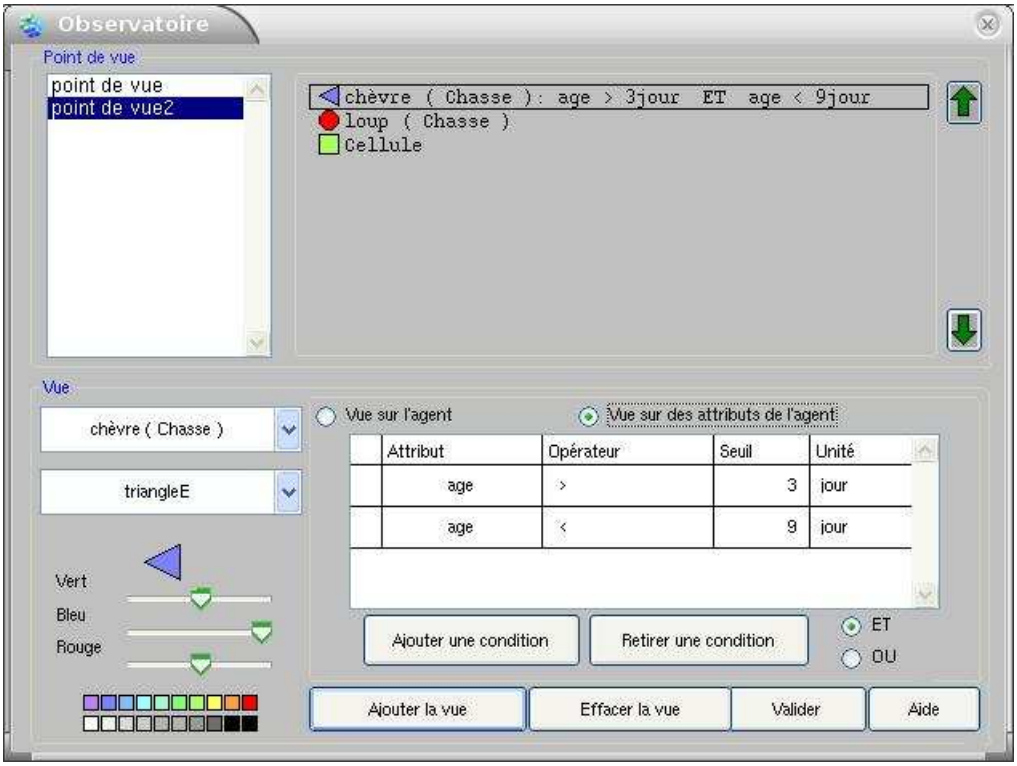

FIG. 2.2: Interface Observatoire pour la création des *Points de vue*.

Quand la liste des objets sélectionnés pour la vue est plus grande que la fenêtre proposée par l'interface, il est possible de naviguer dans la fenêtre

grâces aux flèches haut et bas situées au droite.

Ajouter un point de vue

À partir de la boîte de dialogue appelée par la commande *Observatoire* la procédure pour créer un nouveau *point de vue* est la suivante :

- Dans la boite en haut à gauche **(Point de vue)**, sélectionner grâce au menu contextuel *créer un point de vue* et lui donner un nom. Ce même menu permet également de dupliquer ou de supprimer un *point de vue*, ainsi que de sauver ou de rappeler un ensemble de points de vue dans un fichier binaire suffixé **.pdv**.
- Choisir l'agent à observer parmi la liste proposée dans la comboBox *Vue*.
- Fixer une couleur, prédéfinie ou en jouant sur les curseurs, couleur ou N&B.
- Pour un agent situé, fixer une forme (comboBox du bas). La vue peut porter sur l'agent lui même (sans conditions) ou bien sur son état, donc sur des conditions sur ses attributs.
- En cas de vue sur l'agent (choix par défaut) cliquer directement sur le bouton *Enregistrer la vue*.
- En cas de vue sur des attributs de l'agent, cocher la case *vue sur des attributs de l'agent*. Une boite de conditions apparaît. Il est possible de mettre plusieurs conditions de types *et* & *ou*.
- Quand tout est correct cliquer *Enregistrer la vue*. La vue s'affiche dans la liste en haut à droite (flèches à droite pour faire défiler cette liste si les vues sont nombreuses).
- Quitter la fenêtre par *Valider*. Le point de vue est immédiatement pris en compte si une fenêtre de visualisation est ouverte.

Modifier un point de vue

- Sélectionner le point de vue à modifier dans la boite en haut à gauche **(Point de vue)**, les éléments du point de vue s'affichent à coté. La sélection d'un élément à modifier peut se faire de 2 manières :
  - en cliquant sur l'élément
  - en choisissant l'élément dans la liste de la comboBox *Vue*.
- Ajouter l'élément à la vue avec le bouton *Ajouter à la vue*
- Faire la modification souhaitée
- Supprimer l'ancien élément de la vue avec le bouton *Effacer la vue*
- Valider le nouveau point de vue avec le bouton *Valider*

Remarques

- Pour sélectionner un autre point de vue, rappeler cette fenêtre, rechercher et sélectionner le point de vue souhaité puis valider. Le nouveau point de vue est pris en compte.
- Il n'est pas possible de modifier l'ordre des vues. Or cet ordre peut avoir de l'importance. Les créer dans le bon ordre. Au besoin, on peut sélectionner une vue dans la liste et l'effacer à l'aide du bouton *Effacer la vue*.

2.4.2 Visualisation

Ouvre ou remet en premier plan la fenêtre de visualisation de l'espace. La façon de visualiser les agents se définie avec la fonction **Observatoire**, elle se paramètre avec les préférence du menu **Options**. La fonction **Magnétoscope** utilise également la visualisation pour re-afficher une simulation.

Attention : le rappel au premier plan de la fenêtre de visualisation ne fonctionne pas sous Linux, il faut explicitement ré-ouvrir la fenêtre de visualisation si celle-ci est pliée dans la barre.

2.4.3 Scheduler

La boîte de dialogue appelée **scheduler** (voir la figure 2.3) permet de spécifier l'exécution et l'ordre des principales opérations d'une simulation:

- Exécution et sauvegarde des différents types d'agents,
- Mise à jour de la visualisation.

Elle permet en outre de choisir entre les deux modes d'exécution des agents:

- le *mode asynchrone* (i.e. séquentiel, c'est la valeur par défaut), est le plus simple et le plus sûr. Chaque agent (ordre tiré aléatoirement) joue à tour de rôle et modifie le monde en temps réel.
- le *mode asynchrone* (i.e. parallèle). Chaque agent voit le monde tel qu'il se trouvait au début du pas de temps. L'ordre d'appel n'a donc en principe plus d'importance mais il faut en général gérer d'épineux problèmes de conflits (une même proie ne peut être mangée plusieurs fois !). Ce mode s'applique en priorité aux cellules, en particulier lorsqu'on souhaite les utiliser comme réseau d'automates cellulaires (voir le tutoriel *jeu de la vie* pour un exemple d'automate cellulaire).

le bloc *Liste des opérations* permet d'ordonner la séquence des différentes opérations réalisées par le scheduler. pour cela il suffit de sélectionner une opération donnée et la déplacer dans la colonne grâce aux flèches haut

FIG. 2.3: Interface du scheduler

et bas situées à droite du bloc. Il est également possible de sélectionner/désélectionner certaines opérations en cochant/décochant les opérations concernées.

définir la sauvegarde Le bouton *Définir la sauvegarde* lance la boîte de dialogue qui permet d'affiner la sauvegarde des agents (voir la figure 2.4). Par défaut tout est sauvegardé à chaque pas de temps (sauf si ces opérations sont désactivées dans le scheduler), ce qui peut être extrêmement coûteux en mémoire surtout si beaucoup d'agents naissent et meurent à chaque pas de temps. La fenêtre qui s'affiche permet de sélectionner les agents à sauvegarder, ainsi que leurs attributs. Il est possible également ralentir le rythme de sauvegarde tous les n pas de temps.

Attention pour visualiser les attributs qu'on souhaite sauvegarder il faut auparavant cliquer dans un des moules de la colonne de droite.

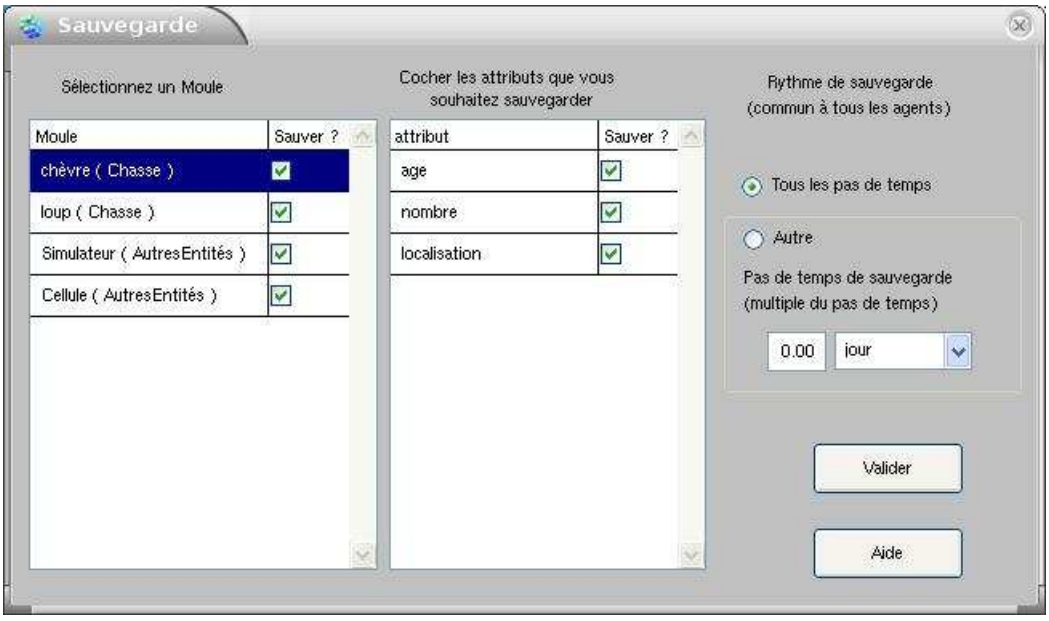

FIG. 2.4: Interface de sauvegarde des résultats

2.4.4 Magnétoscope

Le *Magnétoscope* est une boîte de dialogue dont le fonctionnement est basé sur le concept d'un magnétoscope afin de pouvoir re-visualiser une expérience simulatoire. Il est possible de faire avancer le déroulement de la visualisation pas-à-pas, de revenir en arrière, d'aller directement à la fin etc. Le défilement se fait à la vitesse désirée et une information indique le pas de temps courant par rapport au nombre total de pas de temps.

2.4.5 Chronique

Une chronique est un graphe qui porte le temps en abscisse et la valeur d'un attribut d'un agent ou d'un groupe d'agents en ordonnée. Elle permet de visualiser l'évolution dans le temps d'une des caractéristiques de l'état d'un agent.

Pour ouvrir la boîte de dialogue de visualisation d'une chronique, après avoir exécuter une simulation, il suffit de sélectionner l'item *chronique* du menu *Outil* (ou d'utiliser le raccourci).

Le *gestionnaire des chroniques* permet de définir la chronique que l'on souhaite visualiser, pour cela il faut renseigner les champs de la manière suivante :

- *En préambule* : si le modèle est spatialisé, il faut tout d'abord sélectionner sur la *vue spatiale* la ou les cellules sur lesquelles porte la chronique (par défaut, elle concerne toutes les cellules).
- *Sélectionner* : Dans le cas ou on visualise des animats, il est possible de préciser *tous* (par défaut), ou *seulement les animats toujours vivants en fin de simulation.*
- *Entité/Espèce* : Sélectionner ensuite *l'entité/espèce* à visualiser. Pour les animats il s'agit du nom de l'espèce. Les cellules et les agents non situés sont regroupés sous l'entité *AutresEntités*. Sélectionner *Toutes les espèces* dans le cas ou sont mélangées des espèces d'animats différentes.
- *Agent/Stade* : sélectionner le type d'agent à visualiser, donc le stade pour les animats, la cellule ou l'agent non situé désiré. Pour les animats, il est possible de sélectionner *tous* pour tous les stades.
- *ID* : sélectionner l'identifiant de l'agent dans le cas ou on ne s'intéresse qu' à un unique agent ou *tous* dans le cas de tous les agents sélectionnés par les comboBox précédentes.
- *Attribut* : sélectionner l'attribut à visualiser.

On peut ensuite :

- Visualiser le graphe (bouton *Dessin*)
- Éditer les couples de valeurs (bouton *Texte*)
- Sauvegarder les valeurs de ce graphe dans un fichier à l'aide de la boîte de dialogue de gestion des fichiers (bouton *Sauver texte*)

La figure 2.5 représente le schéma d'une chronique du nombre de chèvres, les valeurs sont listées dans la figure 2.6.

Remarque Le cas ou plusieurs agents sont sélectionnés, Mobidyc fait **la somme** des valeurs de leurs attributs (seule option pour l'instant).

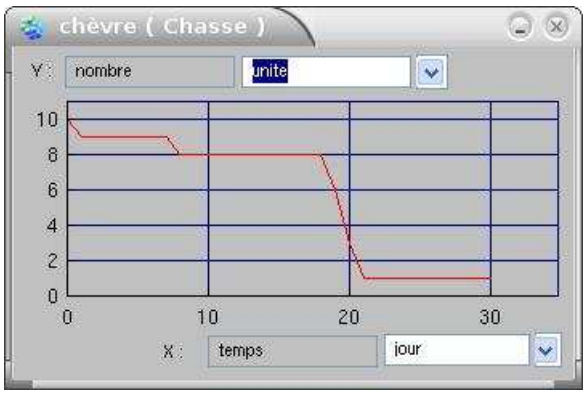

FIG. 2.5: graphe de l'évolution du nombre de chèvres

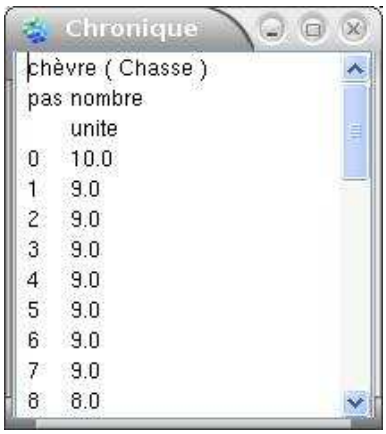

FIG. 2.6: valeurs de la série chronologique *Nb de chèvres*

2.4.6 histogramme

Un histogramme porte en abscisse des intervalles de valeurs pour un attributs, et en ordonnée le nombre d'agents qui appartiennent à ces intervalles de valeurs.

Pour définir un histogramme il faut d'abord, tout comme pour une chronique, définir les agents concernés ainsi que l'attribut à éditer. Si le modèle est spatialisé, il faut d'abord sélectionner sur la vue spatiale la ou les cellules sur lesquelles porte le graphe (par défaut, sur toutes les cellules). Puis il faut remplir les champs de la manière suivante :

- *l'entité espèce* : Pour les animats il s'agit du *nom de l'espèce*. Les cellules et les agents non situés sont regroupés sous l'entité *AutresEntités*. Sélectionner *Toutes les espèces* pour mélanger des espèces d'animats différentes.
- *Agent/Stade* permet de sélectionner le type d'agent à visualiser, donc le stade pour les animats, la cellule ou l'agent non situé désiré. Pour les animats, sélectionner *tous* pour tous les stades.
- *Groupe* fonction désactivée. *tous*.
- *Attribut* permet de sélectionner l'attribut dont on veut éditer l'histogramme. Seuls les attributs communs aux différents agents sélectionnés sont accessibles. Cette liste peut donc être vide.

Il faut ensuite définir l'histogramme à proprement parler :

- Saisir la date (en fait le pas de temps) sur lequel doit porter le graphe
- Saisir le pas de l'histogramme, c'est à dire la largeur des classes. Par exemple un histogramme représentant la distribution en longueur des agents pourra avoir un pas de $10cm$ si la population est comprise entre $0et100cm$(cela correspond à 10 barres maximum).
- Saisir les bornes du graphe, dans la même unité que celle du pas. La valeur -1 (par défaut) veut dire que le graphe s'adapte aux valeurs mini et maxi.

Le graphe est défini par son *pas*, et le nombre de barres dépendra de l'étendue des valeurs (ainsi que des bornes du graphes).

Lorsque le graphe s'affiche, un bouton *animation* permet de visualiser le *film* de l'histogramme dans le temps. Il est alors préférable de donner des bornes fixes à l'histogramme plutôt que de le laisser s'adapter aux valeurs mini et maxi rencontrées à chaque pas de temps.

2.4.7 Tableau de bord

La boîte de dialogue *Tableau de Bord* (voir figure 2.7) permet au modélisateur d'avoir accès de manière pratique à l'ensemble des paramètres qu'il souhaite faire varier. Il se présente sous forme de tableau composé des

colonnes suivantes :

- *Nom du paramètre* est le nom fixé par la boîte de dialogue qui sert à définir celui-ci. Il peut être modifié en cliquant dans le champs.
- *Valeur du paramètre* est la valeur que l'utilisateur a fixé à la création de son modèle. Elle peut être modifiée en cliquant dans le champs.
- *Unité* permet de spécifier l'unité du paramètre choisi (*cliquer dans le champs*)
- *Chemin complet du paramètre* est le nom *complet* que Mobidyc fixe pour ce paramètre donné, c'est celui qu'utilise le moteur pour spécifier ce paramètre. Ce champs n'est pas modifiable, en cliquant dessus on sélectionne toute la ligne pour supprimer le paramètre par exemple.

Des boutons sont disponibles pour manipuler les paramètres du modèles :

- *Ajouter des paramètres* appelle la boîte de dialogue graphique qui représente les paramètres du modèles. Pour sélectionner les paramètres qu'on souhaite afficher dans le tableau de bord voir la procédure d'ajout des paramètres, section 4.1.
- *Ajouter tous les paramètres* ajoute tous les paramètres à l'affichage dans le tableau de bord
- *Supprimer un paramètre* supprime le paramètre sélectionné du tableau de bord
- *Tout supprimer* supprime de l'affichage l'ensemble des paramètres pré-

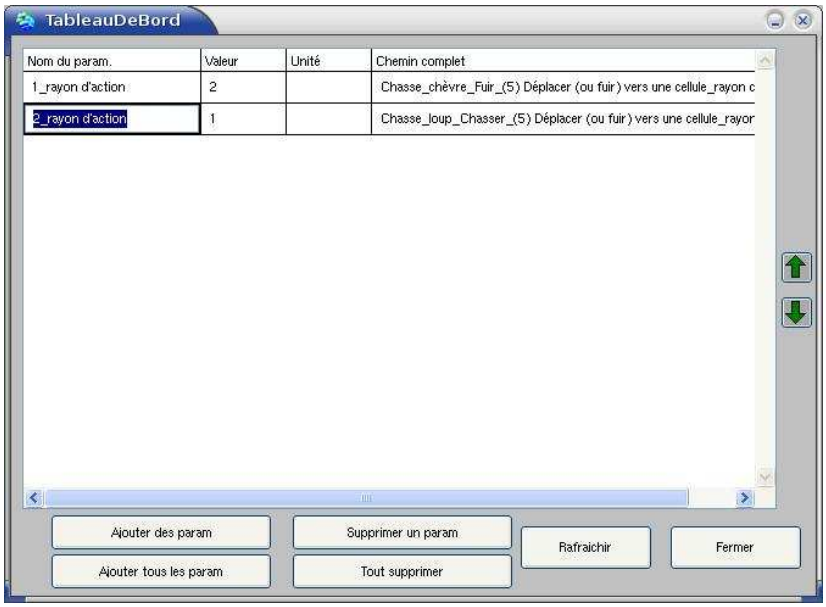

FIG. 2.7: Tableau de bord

sent dans le tableau de bord
- *Rafraîchir* permet de maintenir la cohérence avec le modèle, si par exemple un paramètre a été supprimé par ailleurs.
- *Fermer*

Les flèches situées à droite permettent de changer l'ordre d'affichage des paramètres. Cela peut être très utile dans le cas ou de nombreux paramètres sont affichés afin de visualiser en premier ceux considérés comme le plus important.

Les 3 groupes de commandes qui suivent contiennent les méthodes qui permettent directement la construction explicite d'un modèle.

2.5 Espace

La première étape concrète (après avoir spécifier le *nom* du projet) lors de la création d'un modèle est de *définir* un espace. Ce menu est destiné à la manipulation de ce concept.

2.5.1 Créer une grille

- Il est possible de spécifier la taille de la grille
- La forme des cellules est
  - Carrée à 4 voisins
  - Carrée à 8 voisins
  - Hexagonale
- les frontières sont *fermées* ou *ouvertes* (de type *tore).*
- Il est possible de spécifier la taille de la fenêtre de visualisation, cela revient à fixer une échelle de représentation des cellules.

2.5.2 Définir les cellules

Les agents cellules forment l'environnement. Tout comme certains modèles n'utilisent pas d'animats (ex. les automates cellulaires), les modèles non spatialisés n'utilisent pas d'agent cellules. Mais le système crée alors une cellule unique dans laquelle se trouveront tous les agents. L'utilisateur peut ne pas en tenir compte, ou utiliser cette cellule comme il utiliserait un agent non situé.

Cette boîte de dialogue permet de donner des *attributs*, c'est-à-dire un *état* à l'espace. Cela est possible pour tout ou partie des cellules qui composent l'espace.

Dans le cas ou on désire spécifier une partie de l'espace il faut désigner avec la souris les cellules concernées dans la fenêtre de visualisation, pour cela voir les tableaux 2.1, 2.2 et 2.3 qui récapitulent les règles de fonctionnement de sélection des cellules selon le système d'opération choisi.

Sélectionner	**Linux**
Une cellule	clic Gauche
Toutes les cellules	Clic Droit
Additionner les cellules	shift-clic Gauche
Toutes sauf	ctrl-shift-clic Gauche
Aucune	*toutes les cellules*, puis *Toutes sauf*

TAB. 2.1 – *mode de sélection des cellules de la fenêtre de visualisation sous Linux*

Sélectionner	**Mac**
Une cellule	clic G
Toutes les cellules	Clic D
Additionner les cellules	shift-clic
Toutes sauf	ctrl-shift-clic G
Aucune	*toutes les sélectionner*, puis *Toutes sauf*

TAB. 2.2 – *mode de sélection des cellules de la fenêtre de visualisation sous Mac*

Sélectionner	**Windows**
Une cellule	clic G
Toutes les cellules	Clic D
Additionner les cellules	shift-clic D
Toutes sauf	ctrl-shift-clic G
Aucune	*toutes les selectionner*, puis *Toutes sauf*

TAB. 2.3 – *mode de sélection des cellules de la fenêtre de visualisation sous Windows*

À l'appel de la boîte de dialogue *Définir des cellules*

- Si aucune cellule spécifique n'a été sélectionnée le bloc *Sélection des cellules* affiche *Toutes* (l'autre choix est inhibé)
- Si une partie des cellules a été sélectionnée le bloc *Sélection des cellules* affiche *À la souris* (mais il est possible de choisir *Toutes*).

Par défaut, toutes les cellules de *la sélection des cellules* sont sélectionnées. La comboBox *cellules en cours de modification* donne les identifiants des

cellules sélectionnées, mais il est possible de choisir soit :

- une cellule par son identifiant (grâce à cette comboBox)
- un sélection préétablie à l'aide de la souris

Pour les cellules sélectionnées il est alors possible d'ajouter ou de modifier les *attributs* ou/et *tâches* de celles-ci. Pour cela dans le bloc *La Cellule* il faut :

- *Sélectionner un nouvel attribut ou une nouvelle tâche* dans la liste proposée (dans le cas d'un ajout)
- Sélectionner l'attribut ou la tâche existante dans le tableau *Attributs* ou *tâches* et modifier (ou supprimer) celui-ci avec le clic droit de la souris. On lance ainsi la boîte de dialogue de l'attribut ou de la tâche sélectionné pour faire les modifications demandées.

Il est possible de rendre toute ou partie des tâches inactives en supprimant la coche située à droite du tableau des tâches.

2.5.3 Enregistrer l'espace

cet item permet de conserver de manière pérenne, sous forme de fichier suffixé `.esp` un espace spécifié. Cette commande est automatiquement effectuée lorsqu'on sauve un projet.

2.5.4 Charger un espace

Cet item consiste à charger un fichier `.esp` (de type texte) de description des cellules (taille et type de la grille, et état initial des cellules). Attention, le moule des cellules, donnant en particulier le comportement des cellules, reste dans le fichier `.mbp`. Cette commande est automatiquement effectuée lorsqu'on charge un projet.

2.5.5 Charger un fichier de contours

Cet item permet d'importer un fichier de type `ASCII` de contours de cellules de formes quelconques. Mobidyc calcule automatiquement le voisinage de chaque cellule. Voir le fichier *polygone.cel* fourni avec les exemples qui donne le format à utiliser.
NoteDeHR: Préciser comment spécifier un fichier contours

2.5.6 Exporter les données

Cela permet l'exportation en mode texte des résultats concernant les cellules (menu Espace $->$ exporter les données...). Le résultat est un tableau par attribut, avec les cellules en lignes et les pas de temps en colonne.

2.6 Peuplement

2.6.1 Définir les animats

Les animats sont les agents du peuplement. Ils sont situés dans l'espace (l'attribut *localisation* donne le numéro de leur cellule) et ce sont les seuls qui peuvent se métamorphoser, se reproduire, se déplacer et mourir. Cette boîte de dialogue permet la définition de *l'état* et du *comportement* des *animats*. Dans mobidyc, un *animat* se définit toujours comme le stade d'une entité, voir le tableau 2.4 qui montre plusieurs exemples.

Stade	**Entité**	**Animat**
oeuf	truite	*un oeuf de truite*
adulte	truite	*une truite adulte*
agent	sugar	*un agent sucre*

TAB. 2.4 – *Description des animats sous Mobidyc*

La fenêtre est structurée en deux blocs :

- *Le peuplement* : Ce bloc sert à créer ou sélectionner un animat
- *L'agent* : ce bloc sert a créer ou modifier l'état et le comportement de l'animat sélectionné.

Bloc Peuplement

Ce groupe est composé de deux listes : les *entités* et les *stades*.Pour créer un nouvel animat sélectionner *créer une nouvelle entité* dans le menu contextuel de la liste *Entités* et donner un plusieurs noms de stades. L'entité et les stades s'affichent dans les listes correspondantes.

Entité le menu contextuel de cette liste permet les actions suivantes :

- *Créer* une nouvelle entité avec son stade (obligatoire)
- *Dupliquer* l'entité sélectionnée
- *Supprimer* l'entité sélectionnée
- *Ajouter des stades* à l'entité sélectionnée
- *Vider le peuplement* supprime **toutes** les entités dans le cas d'une réinitialisation du peuplement

Stades le menu contextuel de cette liste permet les actions suivantes :

- *Supprimer* supprime le stade sélectionné
- *Dupliquer* duplique le stade sélectionné (mais il faut lui donner un nouveau nom).

Bloc Agent

Ce groupe permet de définir ou modifier les attributs et le comportement de l'animat précédemment sélectionné.

Sélectionner un nouvel attribut ou une nouvelle tâche Pour ajouter ou modifier l'état et/ou le comportement de **l'animat sélectionné** dans le bloc *Le peuplement* il faut d'abord choisir dans le champs *Sélectionner un nouvel attribut ou une nouvelle tâche* parmi la liste suivante :

- **Attributs :**
  - *Attribut standard*
  - *Attribut de type liste*
  - *Attribut dépendant*
- **Assembler une tâche**
- **Tâches générées**
- **Tâches prédéfinies :**
  - *Vieillir*
  - *Métamorphose*
  - *Reproduction*
  - *Déplacement*
  - *Mourir*
  - *Survie*
  - *Modifier attribut*
  - *Mathscript*
  - *Scénario*
  - *Batch*
  - *Alias*

Ensuite il faut suivre les instructions des boîtes de dialogues correspondantes (voir la section 3.3 pour les détails). Les attributs et les tâches créés apparaissent dans les deux listes du bas. Pour modifier un animat, sélectionner *son entité* et *son stade* puis un attribut ou une tâche préexistants et utiliser les menus contextuels. Les choix seront pris en compte après avoir cliquer sur le bouton *Valider*.

Attention *Il est impossible d'annuler !*

Liste des attributs Ce tableau contient la listes des attributs de **l'animat sélectionné**. Le menu contextuel permet de *modifier* ou *supprimer* l'attribut en question. Lorsqu'on souhaite modifier l'attribut, la boîte de dialogue de description du type d'attribut est appelée (voir la section 3.2).

Liste des tâches Ce tableau contient la listes des tâches de **l'animat sélectionné**. Le menu contextuel permet de :

- *Modifier* une tâche
- *Supprimer* une tâche
- *Éditer l'assemblage* d'une tâche
- *Copier* une tâche
- *Coller* une tâche

Remarques

- Les flèches à droite du *tableau des tâches* servent à modifier l'ordre des tâches. Celles-ci sont toujours effectuées en mode séquentiel (ou asynchrone). **L'ordre des taches peut donc avoir de l'importance**.
- La case à cocher à droite d'une tâche dans la liste *Tâches* sert à activer une tâche. a décocher si on souhaite qu'une tâche ne soit pas exécutée, ce qui peut être utile en phase de mise au point d'un agent. Une fois l'agent définitivement au point, supprimer les tâches inutilisées.

2.6.2 définir les agents non situés

Les agents *non situés* sont des agents optionnels qui ont souvent des fonctions de d'intérêt général ou d'observateurs de la simulation. Ils sont toujours accessibles (par leur nom) par tous les autres agents. *L'agent non situé* peut par exemple porter les scénarios généraux de la simulation (saisons, températures...) ou enregistrer les résultats essentiels d'une simulation.

Cette boîte de dialogue permet de définir ou de modifier un *agent non situé*, pour cela il faut :

- *Créer un nouvel animat*. Ce bouton appelle la boîte de dialogue qui permet de définir un nom à cet animat (les *agents non situés* sont regroupés sous l'entité *AutresEntités*). Le nouvel agent non situé apparaît dans le tableau *Nom* de la boîte de dialogue. À l'aide du menu contextuel il est possible de
  - Créer un nouvel animat non situé : *Nouveau*
  - Supprimer un animat non situé : *Supprimer*
  - Dupliquer un animat non situé :*Dupliquer*
- Il est ensuite possible d'ajouter un ou plusieurs attributs et tâches à l'aide du bouton *Ajouter* après avoir *Sélectionner un nouvel attribut ou une nouvelle tâche* à partir du menu déroulant proposé. La liste est identique à celle disponible dans la boîte de dialogue *définir les animats* (voir la section 2.6.1). Les attributs et les tâches ajoutées apparaissent dans les deux tableaux respectifs des attributs et des tâches. On retrouve pour chacun de ces tableaux les mêmes outils et menus déroulants que pour la boîte de dialogue *définir les animats*.

2.6.3 Placer les agents dans l'espace

C'est la procédure à suivre pour placer les agents dans l'espace, avant de lancer une simulation, ou bien pour ajouter ou supprimer des agents en cours de simulation. la figure 2.8 montre l'aspect de la boîte de dialogue. Au préalable il faut avoir sélectionner les cellules dans lesquelles seront placés les agents. Par défaut, toutes les cellules sont sélectionnées. Il existe deux manière de placer les animats : la méthode *manuelle* et la méthode *automatique*.

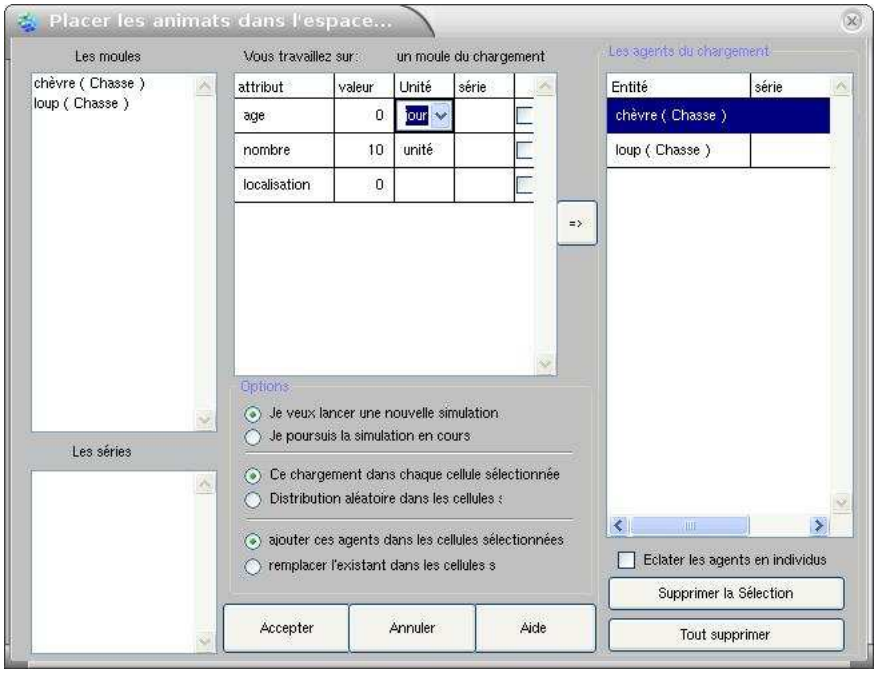

FIG. 2.8: Placer les animats dans l'espace

Création manuelle d'animats (un par un)

Les moules des animats sont listés dans la boîte de gauche. Sélectionner le moule souhaité et créer le *moule du chargement* correspondant en cliquant sur la flèche située entre la boîte centrale et la boîte de droite. L'agent s'affiche dans la boîte de droite *les moules du chargement*.

Sélectionner ce moule du chargement et modifier si nécessaire la valeur initiale des attributs en modifiant les champs *valeurs* de la boîte centrale (par défaut les valeurs du moule seront utilisées comme valeur initiales des attributs de l'agent).

Cette procédure crée les agents un par un. Avant validation finale, il est possible de visualiser/modifier les valeurs des attributs des agents du

chargements en sélectionnant l'agent du chargement désiré. Ses attributs s'affichent alors également dans la boîte centrale, et le titre de la boîte informe que le travail porte bien sur un moule du chargement et non plus sur un moule réel.

Attention dans une procédure de chargement, modifier les valeurs des attributs des moules peut avoir des conséquences fâcheuses. En effet les valeurs par défaut des moules peuvent être utilisées par ailleurs, en particulier pour donner aux agents un état initial lors d'une reproduction.

Pour le placement initial des agents, il est donc en général préférable de modifier les moules du chargement. Ce chargement initial sera sauvé avec le projet.

Création automatique de séries d'animats identiques

Il existe deux procédures :

- Faire comme précédemment puis cocher l'option *Ce chargement dans chaque cellule sélectionnées*. Un agent identique se trouvera dans chaque cellule sélectionnée.
- Faire comme précédemment mais donner une valeur n différente de 1 dans l'attribut *nombre* de l'agent. Cocher ensuite *Éclater les agents en individus*. Il sera créé n agents identiques, puis selon l'option de placement, ces n agents seront répartis aléatoirement dans les cellules (n agents finaux) ou bien seront répétés dans chaque cellules sélectionnées ($n * nb$ cellules sélectionnées agents finaux).

Création automatique de séries d'animats différents

Principe Au lieu de donner une valeur unique aux attributs du moule, on va donner une série de valeurs pour certains attributs. Il sera créé autant d'agents qu'il y a de valeurs dans les séries, le premier agent prenant la première valeur de chaque série, et ainsi de suite. Les séries doivent donc toutes avoir le même nombre de valeur. Les attributs non dotés d'une série gardent leur valeur par défaut.

Procédure Sélectionner un moule d'agent puis sélectionner *créer une série* dans le menu contextuel de la boîte série. Dans la boîte qui s'affiche, sélectionner le premier attribut pour lequel on souhaite des conditions initiales différentes, puis le menu contextuel *Éditer une série* pour créer (ou modifier) la série.

Il est possible de mettre des séries sur autant d'attributs que désirés, *mais ces séries doivent avoir le même nombre de valeurs.*

Un fois validé il faut charger ces séries. Pour cela, sélectionner la série puis faire *charger la série* dans le menu contextuel. La série s'affiche en regard

des attributs concernés dans la boîte centrale. La case à cocher en regard de la série sert à désactiver temporairement une série sur un attribut : tous les agents retrouveront alors leurs valeurs par défaut pour cet attribut.

Attention pour cette procédure, il est préférable de travailler sur le *moule réel*, et non pas sur les *moules du chargement*, pour que les séries soient durablement sauvegardées avec le projet. En effet, les séries ne modifient pas la valeur initiale du moule.

Remarques

- Pour affiner le chargement initial, il est possible de faire plusieurs appels successifs à cette procédure avant de lancer une simulation. Notez les choix *ajouter les agents* ou *remplacer l'existant* dans les cellules sélectionnées. Remplacer et la seule méthode permettant de supprimer des agents. Pour vider les cellules, remplacer par une liste vide d'agents.
- Lorsqu'une simulation est terminée, un appel à cette procédure avec l'option *je veux lancer une nouvelle simulation* réinitialise tout le peuplement (les anciens agents initiaux sont détruits). Si on souhaite lancer une nouvelle simulation en modifiant le placement initial, sélectionner *Programme − > Retour à T zéro* avant d'appeler la procédure de placement.
- Lorsqu'une simulation est terminée, et que l'on souhaite la poursuivre, il est possible d'ajouter ou de supprimer des agents selon la même procédure. Cocher alors *je poursuis la simulation en cours*. Il est également possible de modifier les tâches des agents avant de poursuivre. *Attention, ne pas utiliser cette procédure de modification des agents si la simulation est simplement stoppée. Toujours bien finir un cycle de simulation avant toute modification.*

2.6.4 Charger des moules

Pour rappel, les moules sont des objets qui contiennent la structures des agents, il sont sauvegardés dans un fichier de description suffixé *.ppl*. Cette commande est automatiquement effectuée lorsqu'on charge un projet.

2.6.5 Enregistrer les moules

Cette commande sauvegarde des moules des agents dans un fichier suffixé *.ppl*. Cette commande est automatiquement effectuée lorsqu'on sauve un projet.

2.6.6 Charger un peuplement depuis un fichier

Cette commande appelle un fichier de type texte suffixé *.chg* donnant directement le peuplement initial.

2.7 Tâches

L'ensemble de ce menu permet la création et la modification du comportement de agents. La méthode la plus simple est de créer une tâche par assemblage de *primitives*.

2.7.1 Assemblage ou modification d'un tâche

Un utilisateur peut créer ses propres tâches sans écrire de code, en enchaînant des briques élémentaires appelées *primitives*. La figure 2.9 montre la boîte de dialogue qui est appelée lorsqu'on souhaite créer une nouvelle tâche par assemblage de primitives ou pour modification de celle-ci (voir la section 3.4 pour plus de détail sur les primitives).

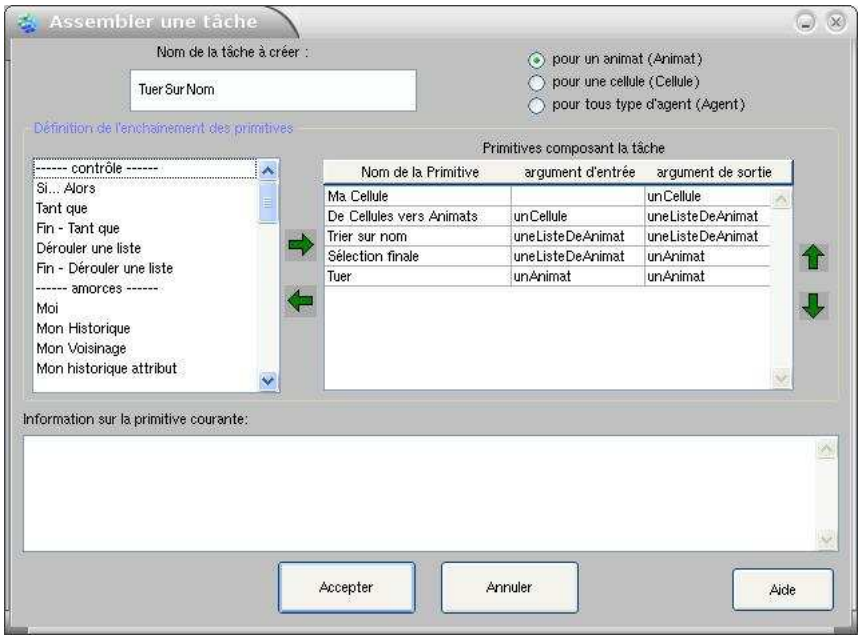

FIG. 2.9: Création d'un tâche par assemblage de primitives

L'assemblage de primitive se fait de la manière suivante :

– Donner un nom à la tâche (sans caractère blanc, et commençant par une majuscule).

- Spécifier à quel type d'agent elle doit servir (par défaut: tous type d'agent).
- Sélectionner les primitives désirées dans la liste de gauche, et les faire passer vers la liste de droite avec la flèche. Les flèches verticales tout à droite servent à modifier l'ordre des primitives de la tâche.
- Valider

Les causes d'erreur les plus fréquentes sont :

- Un oubli de clore la primitive *FinTantQue* ou *FinDérouler*.
- des incompatibilités de type entre arguments: les arguments en entrée d'un tâche doivent être de même type que ceux en sortie de la tâche précédente
- Une erreur sur le propriétaire de la tâche : on ne peut pas tuer des cellules par exemple !

Attention s'il y a une modification du type d'agent propriétaire de la tâche **après** avoir listé les primitives, la modification ne sera pas prise en compte. Il faut donc bien spécifier le propriétaire **avant** de sélectionner les primitives et ne plus le modifier ensuite.

Exemples simples

- Associer la primitive *Conditions* à la tâche *Déplacement* pour créer une tâche *DéplacementConditionnel*
- Associer *Tous les animats*, *Trier sur nom*, *Compter*, pour créer une tâche *CompterCertainsAnimats*

Voir le ***tutoriel*** pour des exemples d'assemblages plus complexes.

Remarques importantes

- Toute primitive acceptant une *liste* en entrée accepte également un *objet unique*.
- Une tâche *assemblée* est selon sa complexité 2 à 6 fois plus lente qu'une tâche *codée*. Un utilisateur averti pourra donc utiliser l'outil d'aide au codage direct des tâches qui nécessite de connaître le langage Smalltalk.
- Si le nom de la tâche créée n'est pas conforme aux règles fixées (pas de caractères espace, pas d'accents) le nom de la tâche est « recalibrée » automatiquement, par exemple *Chèvre Loup* devient *ChevreLoup*.

2.7.2 Création ou modification d'une tâche par codage

S'il connaît Smalltalk, un utilisateur peut coder directement ses tâches dans ce langage.

Le gros intérêt de cette interface est de proposer des accès à du code pré-écrit que l'utilisateur peut insérer dans son propre code. Il trouvera ainsi

immédiatement comment récupérer par exemple l'agent en cours d'activation, tous les agents cellules, leurs historiques, ou encore la cellule occupée par un agent ou inversement. De même pour les fonctions mathématiques prédéfinies, les variables de la simulation etc etc.

Il est également possible de panacher *assemblage* et *codage*. Pour cela, il faut commencer par assembler sa tâche avec des primitives, puis rappeler cette tâche dans cette interface avec le menu *Tâches* $->$ *modifier la méthode exécutée*. Vous pouvez alors compléter la tâche en personnalisant votre code. Il est même possible de rappeler une *tâche codée* pour modification. Un exemple est traité dans le tutoriel.

La procédure pour coder une tâche est la suivante :

- Donner un nom à la tâche (sans caractère blanc, et commençant par une majuscule).
- Spécifier à quel type d'agent elle doit servir (défaut: tous type d'agent).
- Saisir le code dans l'éditeur central. Utiliser le menu contextuel pour copier, coller ou tester des éléments du code (cf. l'environnement VisualWorks). Utiliser la boîte de recherche de code et le bouton *Insérer* pour ajouter des lignes de code prédéfinies.
- Valider.

Remarque importante : *le code prédéfini n'a pas été mis à jour depuis longtemps !!!*

2.7.3 Supprimer une tâche générée

Cette commande appelle une boîte de dialogue qui contient la liste des tâches générées existantes pour le projet. Il suffit de sélectionner la tâche qu'on désire supprimer et de cliquer sur *Supprimer* pour la supprimer.

2.7.4 Créer une formule ou une fonction

Une *formule* est une expression mathématique qui manipule des paramètres. Exemple : $a + b$

Une *fonction* est exactement la même chose sauf qu'un des paramètres portera le nom réservé **x**, qui sera la variable. Exemple : $a + x$
L'éditeur qui permet l'écriture de *formule* ou de *fonction* est donc exactement le même il est basé sur un mécanisme dénommé *ATOLL*.

Attention Avec la version $v2.2$ de Mobidyc, le mécanisme permettant de décrire des expressions mathématiques est changé. Il faut maintenant utiliser la tâche *MathScript* pour construire ces expressions, voir la section 3.3.7 pour une description complète de cette nouvelle tâche.

l'aide ci-dessous est tout de même conservée au cas ou il serait nécessaire de passer par cette méthode de construction de formule ou de fonction. La procédure est la suivante :

- Une fois la boîte de dialogue ouverte il faut tout d'abord donner un nom à cette fonction dans le champs correspondant : Entrer un nom quelconque (car il ne sert pas directement pas la suite). Exemples : AplusB ; Toto1 ; Toto2...
- **ne pas utiliser de symboles** dans le nom (pas de nom du style A+B...)
- **ne pas utiliser les noms** *Formule* ou *Fonction*
- Entrer la formule, **avec un blanc entre chaque item**.
- **mettre des parenthèses partout** (Voir les exemples ci-dessous (section 2.7.4) sous peine de surprise, cet interpréteur ne respecte pas les conventions mathématiques !

Exemples de formules
$a + (b * c)$
$(a + b) * c$
$((a + b) * par)^c$
$3 * (a)^2$

Exemples de fonctions
$(a * x) + b$ $(abscisse * x) + ordonne$

Le tableau 2.5 listes les fonctions mathématiques reconnues par cet éditeur.

sqrt (racine carrée)	**exp**	**log (log décimal)**	**ln (log népérien)**
abs (valeur absolue)	**sin**	**cos**	**tan**
arccos	**arctan**	**arcsin**	

TAB. 2.5 – *Fonctions reconnues par ATOLL*

Remarque : Il est possible, si nécessaire, de vérifier et modifier le code Smalltalk de la formule en éditant la classe générée par cet interpréteur qui porte ce nom. Mais il faut pour cela le code source de Mobidyc et l'environnement Smalltalk complet.

Bugs connus

- Ne reconnaît pas l'inversion de signe $-$.
  $a - b$ fonctionne mais $-a$ n'est pas reconnu, donc il faut entrer $exp(-1 * a)$ au lieu de $exp(-a)$.
- Mobidyc traduit les unités dans son propre système d'unité. Il faut donc faire attention aux fonctions *log* et *exp*. Si par exemple vous faites *log(taille)* avec taille en **mm**, Mobidyc traduit la taille en **mètres** avant de passer au log.

Fonction mathématiques reconnues : sqrt (racine carrée) ; exp ; log (log décimal) ; ln (log népérien) ; abs (valeur absolue) ; sin ; cos ; tan ; arcsin ; arccos ; arctan

2.7.5 Supprimer une fonction ou une formule

Cette boîte de dialogue appelle une liste contenant les fonctions (respectivement les formules), il suffit de sélectionner celle désirée et de cliquer sur supprimer.

Chapitre 3

Les composants les plus courants

3.1 Les éditeurs

Sous ce terme « éditeur » on désigne les boîtes de dialogues que l'on retrouve dans bon nombre d'interface de construction de tâche ou autre et qui permette d'intégrer, soit des *conditions*, soit des *séries* soit des *séries X Y*.

3.1.1 Éditeur de conditions

Une condition est une évaluation du type
`<Variable> Opérateur <ValeurSeuil>`
qui renvoie vrai ou faux.

- *variable* est **obligatoirement** un attribut d'un agent accessible dans le contexte de la condition : agent propriétaire, sa cellule éventuelle, agent non situé, ou gent (proie) que la condition peut éventuellement recevoir en argument. Il faut sélectionner l'attribut désiré à l'aide du menu contextuel.
- *Opérateur* est à sélectionner dans la liste des opérateurs disponibles (> ; < ; estMultipleDe etc.)
- *ValeurSeuil* peut être, selon votre convenance :
  - une valeur fixe, à saisir ainsi que son unité éventuelle dans les champs correspondants
  - la valeur d'un attribut d'un agent. Pour cela, choisir *lien vers un attribut* dans le menu contextuel, puis sélectionner l'agent et l'attribut souhaité.
  - une combinaison (addition ou multiplication) entre la valeur que vous avez spécifiée et celle d'un attribut d'agent. Pour cela, choisir *lien vers une fonction* dans le menu contextuel.

Remarques :

- Il est possible de saisir manuellement le nom d'un attribut plutôt que de le sélectionner. En particulier si cet attribut n'est pas encore défini, ceci évite de quitter cette procédure pour aller compléter le dictionnaire d'attributs de l'agent concerné.
- On revient à une valeur standard en sélectionnant *annuler le lien* dans le menu contextuel de *Valeur2*.
- Vous pouvez combiner des conditions selon les options *et* ou *ou* mais il est impossible de combiner ces deux options dans une même liste de conditions.
- Pour supprimer une condition, sélectionner la condition puis cliquer sur *retirer la condition*.

3.1.2 Éditeur de séries

Cette interface permet de créer ou d'importer une série de valeurs. Cette série sera en général stockée dans un attribut de type *liste de valeurs*. Quatre modes de création sont possibles :

- **Saisie manuelle :** dans la fenêtre de gauche, saisissez au clavier vos valeurs. Une valeur par ligne. Écrasez la valeur *nil* le cas échéant. Le séparateur décimal est le point.
- **Création automatique de valeurs aléatoires :**
  - dans le bloc *tirage aléatoire*, spécifier le nombre de valeurs désirées, puis le type de tirage (réel ou entier) et les bornes du tirage
  - Cliquer sur *Tirage*. La fenêtre de gauche se remplie des valeurs
  - Si on répète l'opération, les nouvelles valeurs s'ajoutent aux anciennes
  - Cliquer sur *Vider* le cas échéant.
- **Importation par copier/coller :** Sélectionner les valeurs (typiquement depuis une feuille de calcul, et - important - en colonne), copier, puis coller dans la fenêtre d'édition en écrasant éventuellement la valeur *nil*.
- **Importation depuis un fichier texte :** dans le bloc *importation/exportation* cliquer sur *charger* et sélectionner le fichier (de type *.npa*) qui contient les valeurs. Ce sont des fichiers texte qui proviennent typiquement d'une feuille de calcul sauvée au format *texte seul*, avec l'extension *.npa*. Pour connaître le format de ces fichiers, le plus simple est d'en créer un : saisir quelques valeurs, donner une unité, cliquer sur *Enregistrer* et donner un nom (sans extension) au fichier. Puis l'ouvrir avec un tableur.

Ne pas oublier de donner une unité aux valeurs si nécessaire, puis valider.

3.1.3 Éditeur de séries X Y

Une série XY est une liste de couples de valeurs. Elle est en général utilisée pour servir de fonction tabulé : connaissant X, on veut la valeur de Y.

Donner un nom à la série, ainsi qu'aux variables X et Y, avec leurs unités éventuelles.

Pour éditer la série, trois solutions (écraser les valeurs nil):

- **Saisie manuelle**. Séparer X et Y par une tabulation, et les couples par un retour chariot. Utiliser le point décimal.
- **Copier/Coller** le tableau de valeurs depuis une feuille de calcul.
- **Importer depuis un fichier**. Pour connaître le format de ce type de fichier, saisir manuellement quelques couples de valeurs puis cliquer sur *Enregistrer*. Le système crée un fichier textuel suffixé *.sxy* qu'on peut éditer et modifier sous un tableur. Toujours bien sauver les fichiers au format *texte seul*, avec l'extension *.sxy*, pour pouvoir les relire sous Mobidyc.

Sélectionner l'option d'interpolation (linéaire ou escalier). Elle sera utilisée au cas où X n'existe pas.

Conseil important :

- Commencer par le couple ($0- > valeur$) pour être sûr d'obtenir une bonne interpolation. C'est indispensable pour les scénarios qui doivent nécessairement avoir une valeur pour $t = 0$.
- Le bouton *Dessin* permet de visualiser la fonction tabulé

3.2 Les attributs

3.2.1 Attribut standard

Cette boîte de dialogue (voir la figure 3.1)permet donner à un agent un nouvel attribut caractérisant son état et dans lequel sera conservée une valeur. Pour cela il faut :

- Entrer un nom (sans blanc et sans caractères spéciaux comme des +, ., * etc.)
- Donner une valeur initiale (seules les valeurs numériques sont actuellement vraiment utilisables...). Cette valeur sera la valeur par défaut de l'agent lors de sa création. Mais elle est susceptible d'évoluer au cours de la simulation si certaines tâches la modifient.
- Si nécessaire, donner une unité à choisir dans la liste.

Attention Les valeurs initiales peuvent être indiquées en Notation scientifique. Ne surtout pas entrer 1/100, mais 0.01, .01, ou $1e-2$

3.2.2 Attribut de type liste

Cet type d'attribut a la même vocation que l'*attribut standard* simplement la différence réside dans le fait qu'il conserve une *liste de valeur* plutôt qu'une valeur simple. Il faut pour cela :

- Entrer un nom (sans blanc et sans caractères spéciaux comme des + . * ...)
- choisir une unité

À cet instant la liste est initialement vide. A chaque fois qu'une tâche modifiera la valeur de cet attribut, la nouvelle valeur sera ajoutée, sans écraser la valeur précédente comme le ferait un attribut standard.

3.2.3 Attribut dépendant

La notion d'*attribut dépendant* correspond à l'idée de lier un attribut au résultat d'une fonction, ce n'est plus la valeur qui est conservée mais cette fonction qui permet de l'obtenir. cependant à ce jour nous avons rencontré aucun cas concret de modèle qui nécessite ce concept aussi ce type d'attribut est en voie d'être supprimé.

3.2.4 Définir une dépendance

Une dépendance est un lien dynamique entre la valeur d'un paramètre (typiquement dans une tâche) et celle d'un attribut d'un agent. La valeur n'est pas possédée par le paramètre mais lorsqu'on demande sa valeur à ce dernier, il va lire cette valeur ailleurs, dans l'attribut d'un agent qu'on lui a spécifié.

FIG. 3.1: Boîte de dialogue pour ajouter un attribut à un animat

Mobidyc propose deux types de dépendance :
- *La dépendance simple* : la valeur du paramètre est remplacée par celle de l'attribut.
- *La dépendance complexe via une fonction* : La valeur renvoyée par le paramètre est la valeur qu'il possède en propre, additionnée ou multipliée par une fonction mathématique de la valeur de l'attribut dont il dépend : $Val = valParam + (ou *)f(valAttribut)$

La dépendance simple C'est une dépendance complexe avec fonction identité et sans tenir compte de la valeur initiale du paramètre. Pour définir une dépendance entre un paramètre et un attribut, appeler le menu contextuel dans le champ de saisie du paramètre (lien vers un attribut, lien vers une fonction, détruire le lien). Dans le cas d'un lien simple, il suffit de sélectionner l'agent et l'attribut sur lequel portera le lien dans la combobox qui s'affiche. (*mon_nomAttribut* ; *maCellule_nomAttribut* ; *son_nomAttribut* etc...).

La dépendance complexe via une fonction
- Spécifier éventuellement la dépendance seule, l'addition ou la multiplication.
- Modifier si nécessaire la valeur initiale du paramètre.
- Sélectionner une fonction mathématique. Si celle-ci n'existe pas il est possible de créer *Nouvelle fonction*. Ce bouton appelle l'interpréteur de fonctions mathématiques (voir l'aide de l'interpréteur à la section 3.3.7).
- Sélectionner l'attribut sur lequel portera la fonction (champ *Abscisse X*)
- Entrer les valeurs numériques des paramètres de la fonction. Comme pour tout paramètre, ces paramètres peuvent eux-mêmes porter un lien (menu contextuel dans le nom du paramètre).
- Pour avoir une idée de l'allure de la fonction, cliquer sur *dessiner la fonction* après avoir spécifié sur quelle échelle (xmin xmax).

Remarque On peut utiliser l'item *fonction tabulée*, cela permet de définir une fonction tabulée, c'est à dire une fonction définie par des couples de points et une méthode d'interpolation entre ces points. C'est souvent très pratique (voir l'aide sur les *série XY* section 3.1.3.

3.3 Les tâches prédéfinies

Les tâches les plus courantes sont implémentées de manière native dans Mobidyc, elles correspondent aux comportements les plus classiques en dynamique des populations. Ces comportements sont décrits de manière basique et rien n'empêche de redéfinir ces tâches en les complexifiant.

3.3.1 Vieillir

Cette tâche incrémente l'attribut spécifié d'une unité de temps à chaque pas de temps. C'est donc un simple compteur. Elle est exactement équivalente à la tâche *ModifierAttribut* dans laquelle il y aurait l'instruction : $mon\_nomAttribut := mon\_nomAttribut + Simulateur\_pasDetemps$

3.3.2 Métamorphose et Reproduction

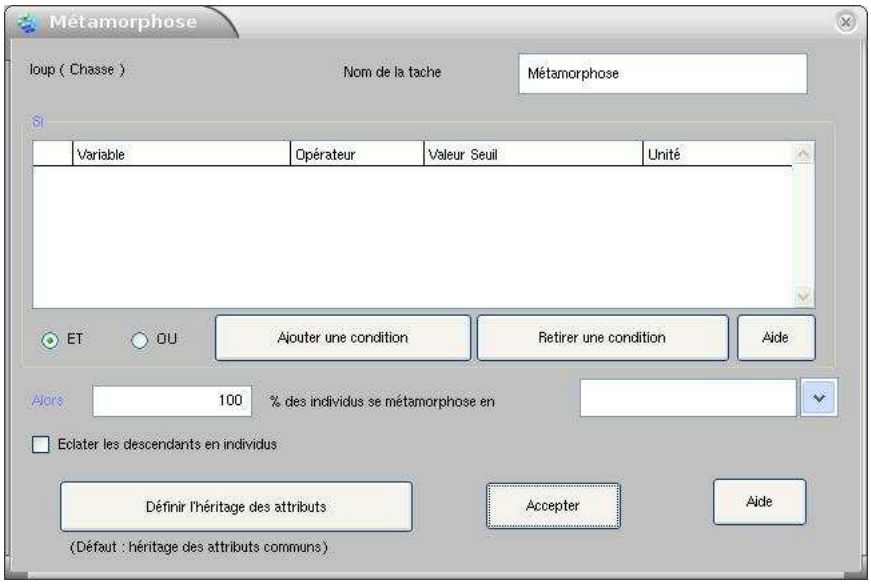

FIG. 3.2: Boîte de dialogue de la tâche *Métamorphose*

La métamorphose (voir la boîte de dialogue de la figure 3.2) permet à un agent de se métamorphoser en son stade suivant. À la suite de cette évolution l'agent parental meurt. La reproduction (voir la boîte de dialogue de la figure 3.3) permet à un agent de créer des descendants et de leur donner des valeurs initiales.

Ces deux tâches partagent de nombreuses particularités, c'est pourquoi leur aide est commune.

- Si besoin, modifier le nom de la tâche, en particulier lorsqu'on souhaite mettre plusieurs fois cette tâche dans un agent. En effet il peut être utile de définir une *métamorphose* ou une *reproduction* vers des types d'agents différents - par exemple mâles et femelles - si vos agents sont sexués.
- Spécifier les conditions de déclenchement de la tâche (par défaut il n'y a aucune condition).

- Spécifier la proportion d'individus qui vont exécuter la tâche (par défaut 100%).
  - Si l'agent représente un individu, cette proportion représente sa probabilité de se métamorphoser ou de se reproduire. Dans le cas de la métamorphose, cela correspond donc à une mortalité de métamorphose (l'agent meurt, même s'il ne s'est pas métamorphosé).
  - Si l'agent représente un groupe d'individu, alors on applique cette probabilité à chaque individu du groupe (avec arrondi aléatoire si nombre non entier d'individus).
- Comme pour (presque) tous les paramètres, il est possible de spécifier une dépendance pour rendre cette proportion dépendante d'un attribut comme la température de la cellule, l'état de l'agent (menu contextuel dans le champ de saisie du paramètre).
- Sélectionner l'agent descendant. Il doit (pour l'instant) appartenir à la même espèce (entité) que l'agent parental.
- Pour la reproduction, spécifier le nombre de descendants par individu (par défaut il y a 1 descendant). Ce peut être un *nombre fixe* (saisir ce nombre), un *nombre à lire dans un attribut* d'un agent (menu contextuel $->$ lien vers un attribut), un *nombre à moduler* (multiplication ou addition) par un attribut d'un agent (menu contextuel $->$ lien vers une fonction), ou un *nombre aléatoire* (entrer les bornes

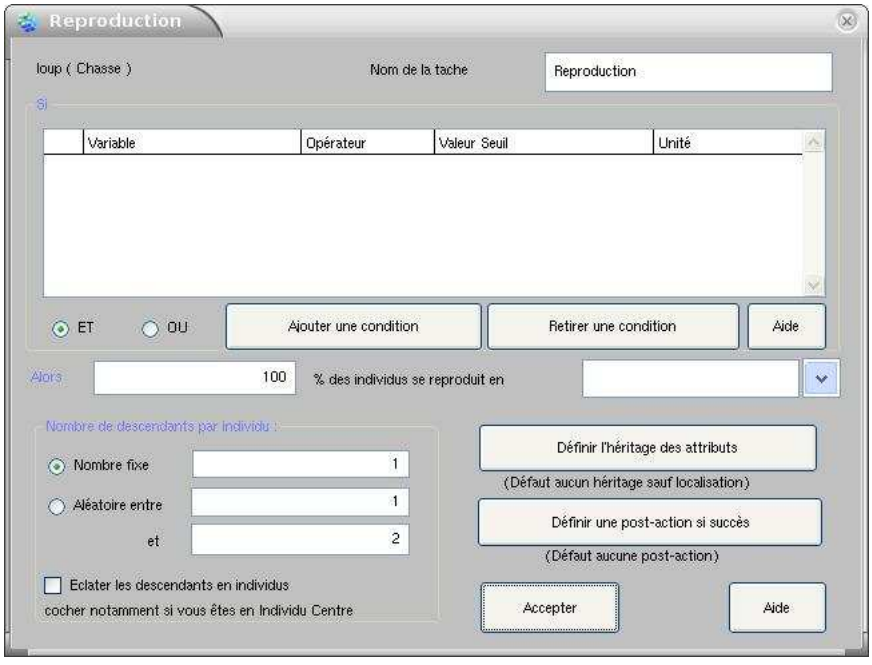

FIG. 3.3: Boîte de dialogue de la tâche *Reproduction*

correspondantes, qui peuvent également porter une dépendance).

- Selon les cas, cocher ou non la case *éclater les descendants en individus*. Si elle est cochée, alors il sera créé autant d'agents unitaires (donc avec attribut *nombre* = 1) que de descendants (soit en gros $nbIndividusParAgent * proba * nbDescendantsParIndividus$ pour la reproduction par exemple). Sinon, il sera créé un seul agent qui groupera tous les descendants (donc avec attribut *nombre* = nbTotalDeDescendants). Il est souvent judicieux de garder les premiers stades (en particulier les stades *œufs* ou *juvéniles* par exemple) groupés pour limiter le nombre total d'agents. Puis on éclate les agents-groupes en individus lors de la métamorphose vers un stade où les individus sont moins nombreux (après les fortes mortalités juvéniles).
- Spécifier l'héritage des valeurs des attributs entre parents et enfants. La valeur par défaut pour la reproduction est *aucun héritage sauf localisation* : les enfants naissent dans la cellule parentale avec comme valeurs initiales les valeurs de leur moule. La valeur par défaut pour la métamorphose est inverse : tous les attributs communs sont hérités. Mais il est souvent intéressant de donner des valeurs qui ne soient ni celles du moule ni celle des parents.

Remarque importante : Dans la fenêtre de définition de l'héritage, si on modifie les valeurs dans la colonne *valeur*, **on modifie la valeur du moule**, c'est à dire la valeur par défaut. Or cette valeur par défaut peut également être utilisée par l'interface de placement initial des animats, ou par une tâche de métamorphose. Pour éviter ce genre de conflit, il est notamment recommandé de modifier les *moules du chargement* plutôt que les moules réels dans une procédure de chargement initial.

Au besoin (pour la reproduction seulement), on peut spécifier une post-action. Une post-action est une tâche qui sera exécutée si les conditions de déclenchement de la reproduction sont remplies. Ceci est valable même si, du fait d'une probabilité différente de 100%, l'agent ne s'est peut-être pas reproduit. Une post-action typique est une tâche de type *MofifierAttribut*, dans laquelle on remet à zéro un paramètre de maturité qui sert à contrôler le déclenchement de la reproduction.

3.3.3 Déplacement

La tâche *Déplacement* permet à un animat de sélectionner une cellule dans un certains rayon d'action et selon différents critères et de s'y rendre **directement** (voir les remarques du paragraphe 3.3.3 pour s'y rendre sur plusieurs pas de temps). Il est également possible d' associer cette tâche à la

primitive *Conditions* (Assemblage de tâches) pour spécifier des déplacements sous conditions. La figure 3.4 représente la boîte de dialogue de cette tâche.

- Si nécessaire, on peut modifier le nom de la tâche pour pouvoir mettre cette tâche plusieurs fois dans un animat.
- Entrer le rayon d'action du déplacement (nombre entier de cellules). Comme pour tout paramètre, il est possible de mettre un lien sur ce rayon, c'est à dire remplacer ou moduler ce paramètre par une valeur lue dans un attribut d'agent (*menu contextuel»lien vers un attribut* et *lien vers une fonction*) voir la section 3.2.4 *définir une dépendance*.
- Définir un filtre sur les cellules. Deux possibilités quil est possible de panacher :
  - Rechercher la (les) cellules dont l'attribut à la valeur la plus forte (gradient ascendant), la plus faible (gradient descendant) ou égale (courbe de niveau) par rapport à la valeur de la cellule sur laquelle se trouve l'animat. Pour cela il faut choisir un attribut de la cellule (comboBox) et le type de gradient.
  - Poser des conditions sur les attributs des cellules (ou d'autres agents). Ces conditions seront testées sur toutes les cellules présentes dans le rayon d'action, et seules les cellules qui vérifient ces conditions seront conservées. Voir le bouton d'aide spécifique des conditions.

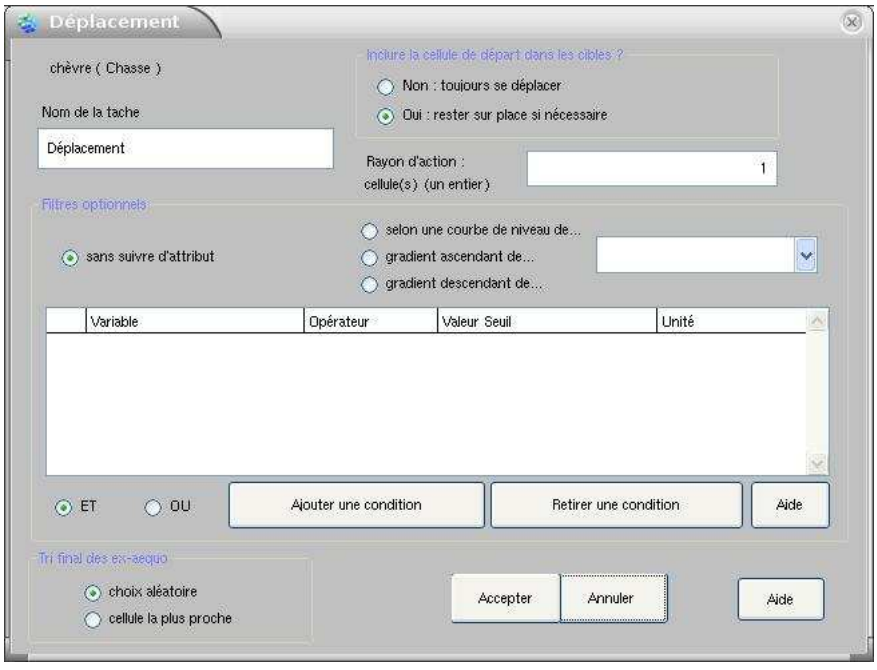

FIG. 3.4: Boîte de dialogue de la tâche *Déplacement*

– Définir le mode de sélection final de la cellule cible : choix aléatoire parmi les cellules candidates ou choix de la cellule la plus proche.

Remarques :

– Dans le cas d'une grille carrée, le résultat sera différent selon qu'elle est définie à 4 ou 8 voisins.

– Il est possible de définir sa propre tâche de déplacement en assemblant des primitives. En particulier, la primitive *déplacerVersUneCellule* permet de spécifier à quelle vitesse on doit s'approcher (ou fuir) de la cellule cible reçue en argument. Un exemple en est donné dans le tutoriel (*chèvre et loup*).

3.3.4 Mourir

La figure 3.5 représente la boîte de dialogue de cette tâche.

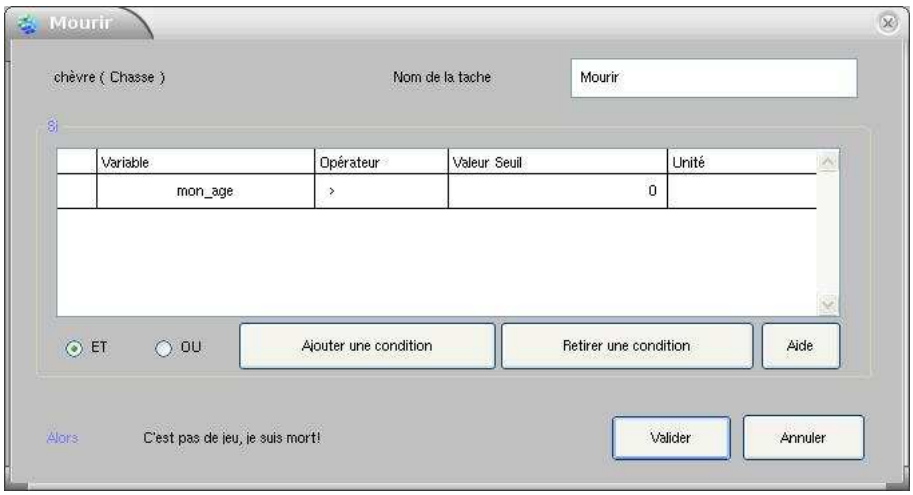

FIG. 3.5: Boîte de dialogue de la tâche *Mourir*

Par défaut cette *tâche prédéfinie* ne possède aucune conditions pour s'exécuter mais il est facile d'ajouter une ou plusieurs conditions grâce à l'éditeur de condition inclus dans cette boîte de dialogue (voir le paragraphe 3.1.1 pour l'aide concernant cet éditeur).

3.3.5 Survie

Cette tâche permet de spécifier une probabilité de survie (fixe) aux animats, à chaque pas de temps.

Deux modes de fonctionnement sont possibles :

– **Le mode discret (et probabiliste)** (mode par défaut) : Le système tire une valeur aléatoire entre 0 et 1, l'agent survie si la valeur tirée

est inférieure à la valeur spécifiée. Si l'agent est composé de plusieurs individus, on répète cette opération pour chaque individu (avec arrondi probabiliste si nombre non entier d'individus), l'attribut *nombre* de l'agent est mis à jour. Si $nombre = 0$ l'agent-groupe est mort.

– **Le mode continu :** Il est à utiliser uniquement pour des agents représentant des populations, donc pour lesquels *nombre* est un réel. On peut fixer la valeur en dessous de laquelle on considère que l'agent-population est mort.

Remarque importante : il n'est actuellement pas possible de mettre une dépendance sur le paramètre de survie. Celui-ci est fixe.

Voir également la tâche Mourir.

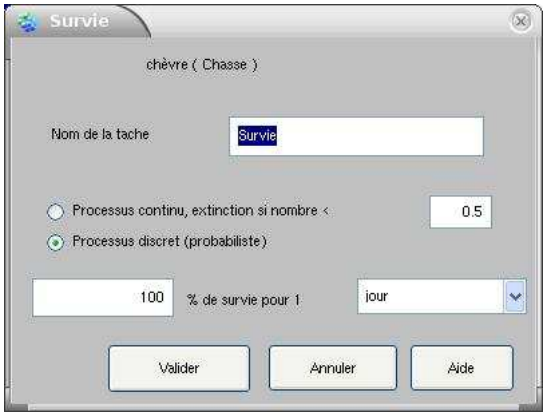

FIG. 3.6: Boîte de dialogue de la tâche *Survie*

3.3.6 Modifier attribut

Tâche d'édition de calculs mathématiques impliquant les valeurs des attributs des agents accessibles dans le contexte de cette tâche. Si par exemple cette tâche est destinée à un animat, alors elle a accès aux attributs de l'animat en cours d'exécution (*mon_nomAttribut*), aux attributs de sa cellule (*maCellule_nomAttribut*), et à ceux de tous les agents non situés (*nomAgentNonSitué_nomAttribut*).

– Modifier éventuellement le nom de la tâche.
– En haut à droite, sélectionner l'attribut récepteur du calcul. Pour éviter les erreurs, il est recommandé que cet attribut préexiste, (il est possible de saisir à la main un nouvel attribut). Ne pas oublier de l'ajouter ensuite à l'agent sous peine d'erreur à l'exécution.
– créer une nouvelle formule mathématique ou sélectionner une existante

- Donner une valeur numérique (unité facultative) aux paramètres, ou utiliser le menu contextuel pour lier ce paramètre à la valeur d'un attribut d'un agent (lien direct ou à travers une fonction mathématique).
- Ajouter la nouvelle expression mathématique à la liste du bas (bouton).
- Si nécessaire, modifier l'ordre des instructions avec les flèches, et éventuellement supprimer certaines d'entre elles grâce u menu contextuel.

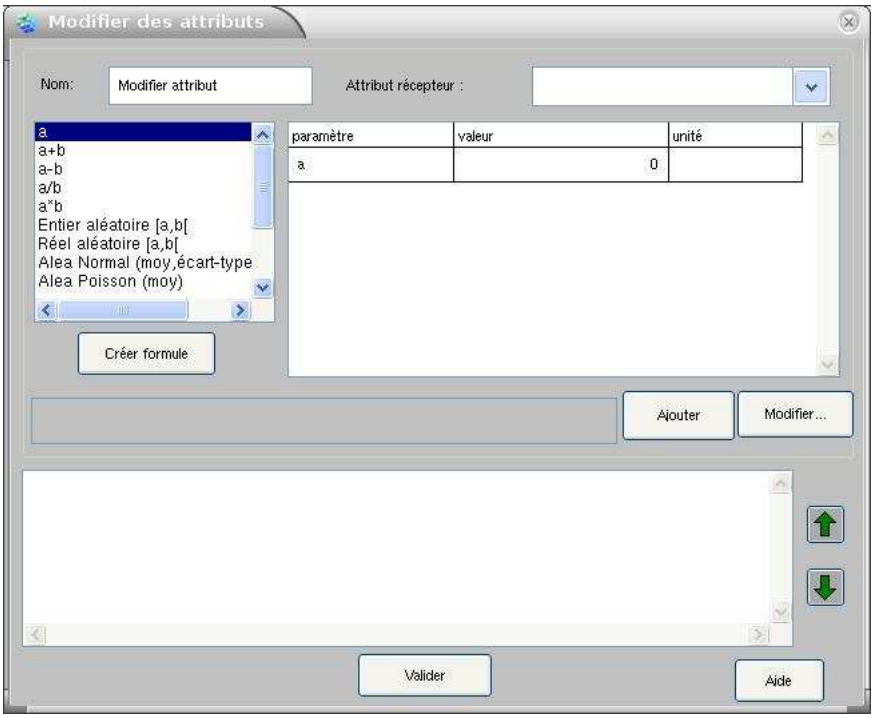

FIG. 3.7: Boîte de dialogue de la tâche *Modifier attribut*

3.3.7 Mathscript

Mathscript est un éditeur de formules mathématiques utilisant un formalisme XML basé sur MathML. Il nécessite Java Runtime Environment (Pour plus de détail voir l'annexe 2.3 sur l'installation de Mathscript).

Mathscript est une interface qui exploite le travail de L. Dirat réalisé dans le cadre de sa thèse [16]. Son travail consistait en partie à livrer un ensemble d'API[1] pour manipuler les outils de visualisation des fonctions mathématiques, la description des fonctions s'appuie sur le formalisme MathML du consortium W3C[2].

1. Application Program Interface
2. voir http://www.w3.org/Math/

La figure 3.8 décrit la boîte de dialogue de mathscript. Elle est composée principalement d'une fenêtre de visualisation des fonctions mathématiques et d'une fenêtre d'édition dans laquelle l'utilisateur écrit ses fonctions, soit directement à la main, soit en utilisant le menu contextuel. la procédure est la suivante :

- Donner un nom (unique) à la tâche.
- Cliquer sur l'onglet *saisie* de la fenêtre d'édition.
- Cliquer sur le clic droit de la souris pour dérouler le menu contextuel,
- Construire son expression mathématique
- Générer la tâche à l'aide du bouton *Générer*, si une erreur de syntaxe ou autre est levée un message apparaît dans une fenêtre *pop-pup*.

Dans cet éditeur il existe 4 index accessibles (dans chaque index, on peut visualiser les pages en les faisant défiler avec les boutons *Précédent* et *Suivant*) :

- *Saisie* la fenêtre d'édition des formules
- *Aide* contient l'aide en ligne
- *Exemples*
- *Fonction tabulé*

Zone d'affichage :

La zone d'affichage permet d'avoir, en temps réel, un aperçu de la formule que l'on est en train de composer. Ce rendu est très proche ce celui qu'aurait une personne écrivant la même formule sur une feuille de papier. Grâce à cet affichage, il est plus aisé de vérifier la cohérence de la formule ou de détecter des erreurs. En effet, la plupart des caractéristiques graphiques mathématiques sont mises en évidence : quotients, inclusions de fonctions, exposants, indices etc.

Zone de saisie :

Cette partie est la partie principale pour l'utilisateur. Elle est accessible via le premier onglet, baptisé *Saisie*. Elle permet de définir le contenu d'un composant MathScript. Un composant MathScript peut contenir les éléments suivants :

- **les formules mathématiques :** l'éditeur permet de saisir plusieurs formules mathématiques, les unes à la suite des autres. Il est possible d'insérer des espaces, de façon à faciliter la lisibilité, mais la convention est de saisir une formule par ligne.
- **les attributs d'agents :** Les agents, et à fortiori leurs attributs, pouvant être nombreux, et afin de faciliter une écriture sans erreur, un menu contextuel est mis à disposition de l'utilisateur. Ce menu s'affiche suite à un clic droit de la souris et liste les agents de la simulation en cours, ainsi que leurs attributs respectifs.

- **les structures de contrôles** : elles permettent de réaliser des tests sur des valeurs de variables ou d'attributs, afin de conditionner le déroulement des calculs. Ces tests sont du type *IF* et *WHILE*. Il est possible de tester si la condition est vraie ou fausse. Les structures de contrôle n'ont pas de contrainte de casse cependant elles sont soumises à une structure bien spécifique. Afin d'éviter toute erreur de structure lors de la saisie et pour faciliter cette dernière, les structures de contrôle sont également accessibles via le menu contextuel.
- **les fonctions :** toujours via le menu contextuel, l'utilisateur peut avoir accès aux fonctions usuelles (trigonométriques, logarithmiques etc.), aux fonctions statistiques (minimum, maximum etc.), aux fonctions proposées par Mobidyc et plus spécifiques à la dynamique des populations (Michaélienne, Bertalanffy ...) ainsi qu'aux fonctions que l'utilisateur a lui-même définies.
- **les paramètres :** ils sont définis par le mot clé *PARAM* (pas de contrainte de casse). L'usage d'utilisation est le suivant :
  $paramnomParametre = valeurParametre(unite)$. En fin de ligne, la description de l'unité est facultative.
- les variables temporaires : elles sont faites pour stocker des valeurs temporaires dans la tâche. Par exemple, on peut y stocker un résultat intermédiaire ou une partie de formule que l'on veut décomposer afin de la rendre plus lisible.
- les commentaires : il est possible d'inclure deux types de commentaires, sur le modèle de commentaire C++ ou java. Le commentaire unitaire suit deux caractères back slash et se termine avec la fin de la ligne. Un bloc de commentaire peut être inséré entre les caractères $/*$ et $*/$

Règles de saisie

Certains caractères sont interdits à la saisie. C'est notamment le cas des caractères accolades { et }, des crochets [et] ou encore du point virgule « ;». Si l'utilisateur tape un caractère interdit, une fenêtre de dialogue munie d'un icône *Stop* s'ouvre et affiche le caractère interdit. Les contraintes d'écriture dans cette zone de saisie sont très restreintes :

- les nombres à virgules s'écrivent avec un point. Par exemple : $pi = 3.14$
- la virgule permet de séparer les paramètres d'une fonction
- les structures de contrôle respectent les structures suivantes :
  - une structure de contrôle commence et se termine par un mot réservé (voir les tableaux 3.1 et 3.2 pour la liste des structures de contrôle).
  - le mot réservé terminant la structure de contrôle est constitué du mot *End* auquel on concatène le mot réservé ouvrant la structure de contrôle. Par exemple : *IfTrue* et *EndIfTrue*.

– Une structure de contrôle comprend toujours une condition. Il est possible de combiner les tests avec les opérateurs logiques *OR* et *AND* (pas de contrainte de casse). Par exemple :

```
IfTrue( ( x<2 OR x>3) AND y = 4)
// Attention de bien structurer les parenthèses
//  pour ne pas avoir d'erreurs de priorité d'opérateur.
```

– Pour les opérateurs de test, il est possible d'inclure une condition complémentaire *Sinon*. Dans ce cas on utilise le mot clé *Else*. Attention, ce mot doit être seul sur la ligne. On concatène le mot *Else* à la fin du mot réservé pour clôturer la structure de contrôle.

ifTrue (condition vraie)	**ifFalse (condition fausse)**
EndIfTrue	**EndIfFalse**
ifTrue (condition vraie)	**ifFalse (condition fausse)**
Else	**Else**
EndIfTrueElse	**EndIfFalseElse**

TAB. 3.1 – *Structure de contrôle :* ***Les tests***

WhileTrue (condition)	**WhileFalse (condition)**
EndWhileTrue	**EndWhileFalse**

TAB. 3.2 – *Structure de contrôle :* ***Les boucles***

Important :

– Dans la fenêtre de visualisation n'est affiché que l'expression mathématique de la ligne courante.

– Les lignes qui contiennent des structures de contrôle ne sont pas visualisée

– un bug important à été identifié et n'est pas corrigé à ce jour : si les noms des attributs contiennent la chaîne *and* ou *or* l'analyseur de mathscript identifie cela comme un *opérateur* de contrôle, d'où la panique qui s'en suit ...

– Le bouton *Générer* permet de valider l'ensemble de la saisie. Cette validation passe par une vérification globale du contenu de l'éditeur. Le système vérifie que les parenthèses sont complètes, que les structures de contrôle sont correctement saisies et que les variables sont bien initialisées. En cas d'invalidité, un message averti l'utilisateur des erreurs. Dans cette fenêtre de message, l'utilisateur peut choisir de valider tout même sa saisie ou d'annuler la validation pour corriger les erreurs. Attention, dans le premier cas, la validation d'erreurs peut entraîner des problèmes dans la suite de la simulation. Il est vivement conseillé de

corriger toutes les erreurs avant la validation finale. Cette validation clôt l'application de saisie des formules.

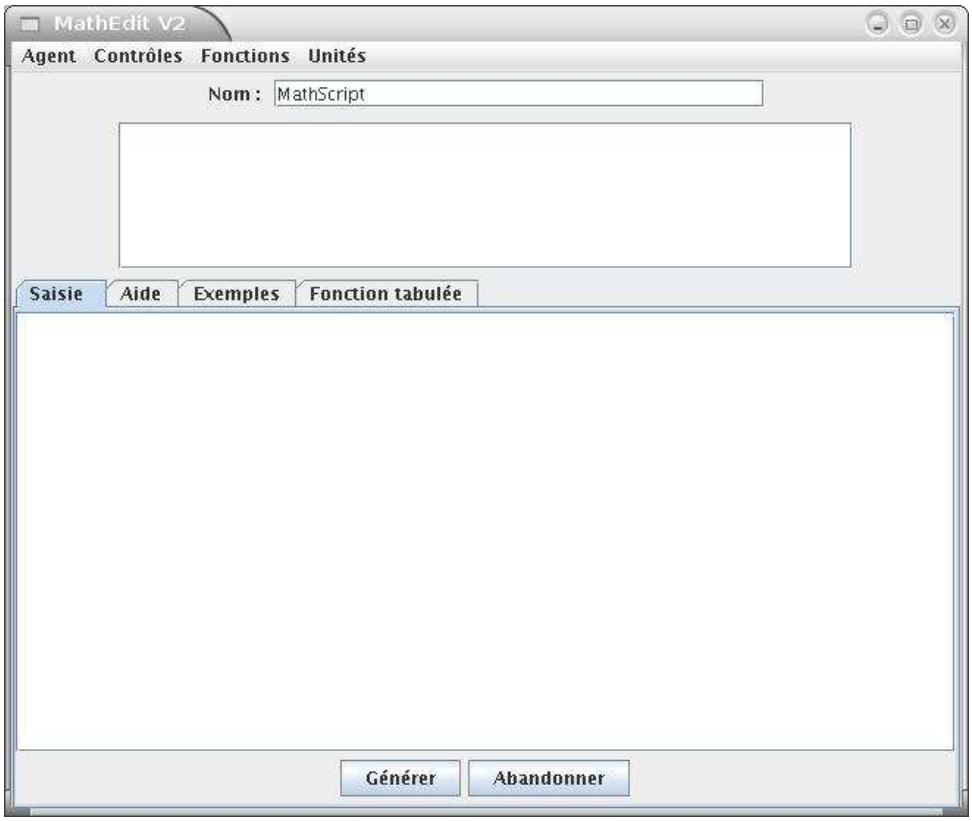

FIG. 3.8: Boîte de dialogue de la tâche *Mathscript*

Exemples :

```
IFTRUE(mon$nbVivants=3 )
mon$statut=1
ENDIFTRUE


IFTRUE( (mon$nbVivants > 3) OR ( mon$nbVivants < 2 ) )
mon$statut = 0
ENDIFTRUE
// exemple 3
IFTRUE(mon$nbVivants=3 )
mon$statut=1
ENDIFTRUE
```

```
IFTRUE( (mon$nbVivants > 3) OR ( mon$nbVivants < 2 ) )
mon$statut = 0
ENDIFTRUE
// exemple 3
IFTRUE(mon$nbVivants=3 )
mon$statut=1
ENDIFTRUE
```

```
IFTRUE( (mon$nbVivants > 3) OR ( mon$nbVivants < 2 ) )
mon$statut = 0
ENDIFTRUE
```

3.3.8 Scénario

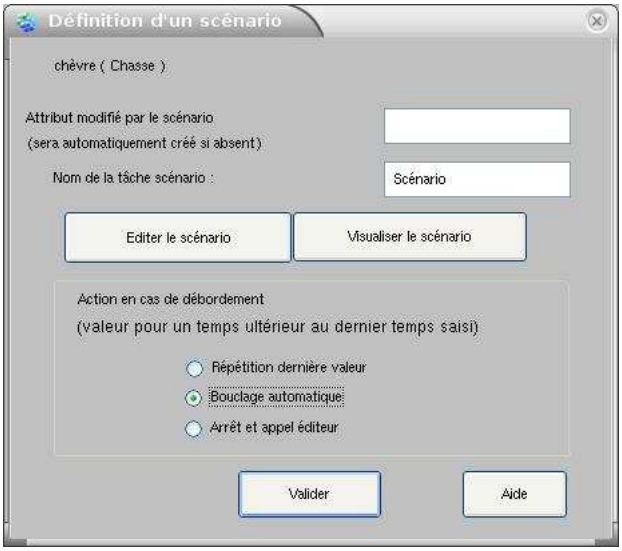

FIG. 3.9: Boîte de dialogue de la tâche *Scénario*

Un *scénario* est un ensemble de couples de valeurs de type *date*$-$ $>$ *valeur*. Cette tâche met à jour l'attribut spécifié en allant lire la valeur correspondant au pas de temps courant dans le scénario. Si l'atribut n'existe pas, il est automatiquement créé.

Cliquer sur *Editer le scénario* et voir l'aide spécifique à la fenêtre d'édition de séries XY (section 3.1.3). Pour un scénario, X représente donc le temps, et Y la valeur de l'attribut.

3.3.9 Scénario dans attribut

Tâche accessible aux **cellules** uniquement.

Même principe que la tâche *scénario*, mais le scénario n'est pas porté par la tâche, il est lu dans un attribut de la cellule, qui doit donc être un attribut de type *sérieXY*.

Si les attributs n'existent pas (y compris celui qui porte le scénario), ils seront automatiquement créés.

3.3.10 Batch

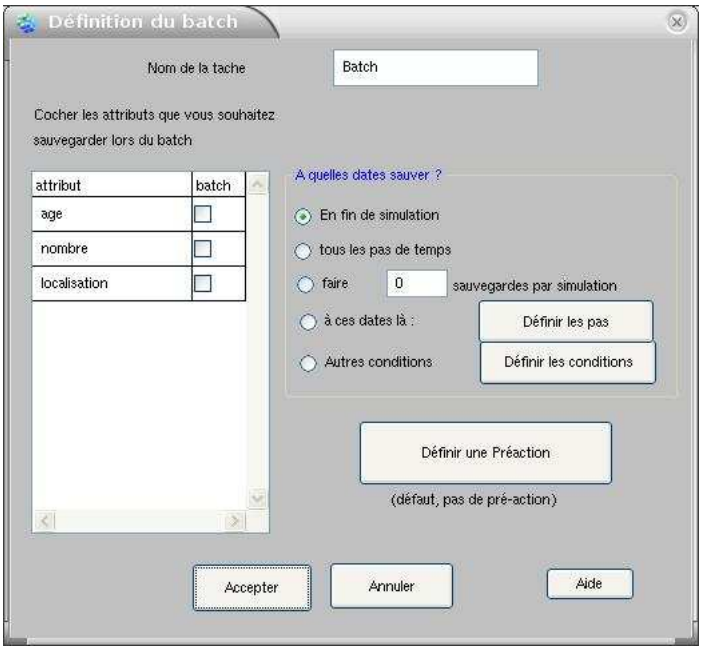

FIG. 3.10: Boîte de dialogue de la tâche *Batch*

Cette tâche, à associer à une expérience simulatoire, permet de sauvegarder les attributs de celui qui la porte pour toute une série de simulations.

- Cocher les attributs qu'on souhaite conserver. Ces attributs peuvent être de type standard ou de type liste.
- Définir à quels instants de la simulation faire les sauvegardes. Par défaut, elle est faite seulement en fin de simulation (au dernier pas de temps).
-

Le bouton: *Définir une Préaction* permet de spécifier une tâche qui sera exécutée juste avant la sauvegarde. Cette option est donc toute indiquée pour effectuer les calculs à mémoriser lors des expériences simulatoires.

Remarques importantes :

- Les animats ne peuvent pas porter cette tâche, elle est réservée aux agents non situés ou éventuellement aux cellules. Il est créé autant d'agents batchs que de porteurs de cette tâche. Mais il est déconseillé de trop les multiplier.
- Cette procédure crée un (ou plusieurs) *Agent Batch*, qui a la même structure qu'un *HistoriqueAgent*, mais le temps est remplacé par le numéro de la simulation. On peut visualiser cet (ces) agent(s) avec l'outil *chronique*, sachant que le temps représente les différentes répétitions des simulations.

3.3.11 Alias

Cette tâche permet d'activer, pour le compte de l'agent qui la porte, une tâche présente dans un autre agent. Elle doit donc être utilisée si deux types d'agents différents partagent exactement la même tâche, par exemple un juvénile et un adulte d'une même espèce, . Ceci évite de paramétrer une même tâche à deux endroits différents.

Il suffit de donner un nom à la tâche, puis de sélectionner la tâche cible.

FIG. 3.11: Boîte de dialogue de la tâche *Alias*

3.4 Les Primitives

Le section 2.7.1 décrit la méthode qui permet de construire une tâche par assemblage de *primitives*, cette section passe en revue l'ensemble des boîtes de dialogue de ces primitives.

Les primitives sont réparties en 6 groupes :

- *les primitives amorces (ou de recherche)*, qui sont en principe en début de tâche (*Moi*, *tousLesAnimats* etc.)
- *Les primitives de sélection*

- *les primitives de traduction*, qui forment le coeur de la tâche et qui ont un argument en entrée et en sortie.
- *les primitives de clôture*, qui sont le plus souvent en fin de tâche (avec ou sans argument de sortie).

À celles-ci s'ajoutent :

- les tâches prédéfinies de Mobidyc
- les tâches crées par l'utilisateur

3.4.1 Primitives d'amorce ou de recherche

Elles caractérisent le point de départ d'une tâche. Elles ne prennent pas d'argument en entrée et renvoie en sortie des agents (réels ou mémoires) ou un *tableau3D* contenant les valeurs mémorisées dans le temps d'un attribut.

Moi L'agent auquel appartient cette tache.
Argument d'entrée : aucun
Argument de sortie : un agent dont le type dépend du propriétaire sélectionné par le radio bouton.

MaCellule Renvoie la cellule sur laquelle se trouve l'animat.
Argument d'entrée : aucun
Argument de sortie : une cellule

TousLesAnimats Renvoie l'ensemble des animats au pas de temps actuel.
Argument d'entrée : aucun
Argument de sortie : une liste d'animats

ToutesLesCellules Renvoie l'ensemble des cellules.
Argument d'entrée : aucun
Argument de sortie : une liste de cellules

MonVoisinage Renvoie les cellules voisines selon un certain rayon. Le propriétaire de la tâche (moi) peut être un animat ou une cellule, mais pas un agent non situé.
Argument d'entrée : aucun
*Argument de sortie :*une liste de cellules

MonHistorique Renvoie l'HistoriqueAgent (la mémoire) de l'agent en cours.
Argument d'entrée : aucun
Argument de sortie : un HistoriqueAgent

MonHistoriqueAttribut Renvoie un *tableau 3D* donnant les valeurs dans le temps (si sauvegardées) d'un des attributs de l'agent. Permet également de sélectionner sur quelle plage de temps on souhaite les valeurs. Fonctionne pour tous types d'agent.
Argument d'entrée : aucun
Argument de sortie : unTableau3D

3.4.2 Primitives de séléction

Parmi la liste des agents fournis en entrée, on en sélectionne certains (ou un seul) sur le nom, sur des conditions à satisfaire, ou bien aléatoirement.

TrierSurNom À partir d'une liste d'animats, permet de sélectionner ceux qui portent le bon nom.
Argument d'entrée : une liste d'animats
Argument de sortie : une liste plus courte d'animats

Trier sur les valeurs des attributs À partir d'une liste d'agents (tous types, y compris historiques), permet de sélectionner ceux dont les *attributs* remplissent des conditions.
Argument d'entrée : une liste d'agents
Argument de sortie : une liste plus courte d'agents

SousSélectionAléatoire Selectionne aléatoirement n objets de la liste d'entrée. Avec n = nombre absolu (entier ou réel avec tirage aléatoire pour la fraction) ou bien proportion (c'est à dire entre 0 et 1). Si un seul élément dans la liste, il sera gardé ou non selon la fraction aléatoire ou la proportion.
Argument d'entrée : une liste d'objets ou un objet
Argument de sortie : une liste plus courte d'objets ou nil

ChoixFinal Choisi un élément de la liste d'entrée selon des critères sur les attributs.
Argument d'entrée : une liste d'animats ou de cellules
Argument de sortie : un animat ou une cellule.

3.4.3 Primitive de traduction

Permet de passer des animats à leurs cellules ou réciproquement de passer des agents aux *historiques* ou au *tableau3D* ou encore de passer d'une cellule à son voisinage ...

DeAnimatsVersCellules Renvoie les cellules sur lesquelles se trouvent les animats listés en entrée.
Argument d'entrée : un animat ou une liste d'animat
Argument de sortie : une liste de cellules.

DeCellulesVersAnimats Renvoie tous les animats qui se trouvent sur les cellules d'entrée.
Argument d'entrée : une cellule ou une liste de cellules
Argument de sortie : une liste d'animats.

DeAgentsVersValeurs
Argument d'entrée : une liste d'animats ou de cellules
Argument de sortie : un *tableau3D*

Voisinage Renvoie le voisinage d'une cellule, selon un certain rayon.
Argument d'entrée : une cellule
Argument de sortie : une liste de cellules

3.4.4 Primitive de calculs

permet de compter les objets reçus en entrée, réaliser des calculs mathématiques impliquant des attributs d'agents, réaliser des calculs simples (somme, moyenne, max, min ...) sur un tableau3D ...
Argument d'entrée : Non commenté
Argument de sortie : Non commenté

Compter Stocke dans un attribut du propriétaire de la tâche le nombre d'éléments non nil contenus dans le paramètre d'entrée. Attention, s'il s'agit d'animats, on obtiendra le nombre d'animats mais pas le nombre d'individus si ces animats représentent des groupes (i.e. si $nombre > 1$).
Argument d'entrée : une liste
Argument de sortie : la liste inchangée

ModifierAttributs Cette primitive permet d'écrire des expressions mathématiques impliquant les attributs des agents.
Exemple type : $mon\_poids := mon\_poids + 0.2 * son\_poids$
avec *son* représentant l'agent passé en entrée. En principe, on fournit un seul agent en entrée. Si plusieurs agents sont fournis, alors la primitive boucle sur tous les agents pris un par un dans l'ordre de la liste. Cette primitive accepte tous types d'agents en entrée dont l'accès se fait via la chaîne «*son_ nomAttribut*» dans les menus contextuels.
Argument d'entrée : un (ou une liste d') agent(s)
Argument de sortie : un (ou une liste d') agent(s)

Remarques : Il faut passer une liste d'agents en entrée, alors la primitive boucle sur tous les agents un par un. Cette méthode peut éviter d'utiliser la primitive de contrôle *DéroulerLesEntrées*.

- Modifier le nom de la primitive (facultatif).
- En haut à droite, sélectionner l'attribut récepteur du calcul. Pour éviter les erreurs, il est recommandé que cet attribut préexiste, cependant on peut saisir à la main un nouvel attribut. Ne pas oublier de l'ajouter ensuite à un agent sous peine d'erreur à l'exécution.
- Sélectionner ou créer une formule mathématique.
- Donner une valeur numérique (unité facultative) aux paramètres, ou utiliser le menu contextuel pour lier ce paramètre à la valeur d'un attribut d'un agent (lien direct ou à travers une fonction mathématique). *Son* fait référence à l'agent passé en argument à la primitive.
- Ajouter l'expression mathématique à la liste du bas (bouton).
- Si nécessaire, modifier l'ordre des instruction avec les flèches avec le menu contextuel.

CalculSurT3D Permet de calculer la somme, la moyenne, le maximum ou le minimum sur une des trois dimensions d'un *tableau 3D.*
Argument d'entrée : un T3D
Argument de sortie : un T3D avec une dimension de moins

MathScript Voir la section 3.3.7 pour connaître en détail cette tâche/primitive.

3.4.5 Primitives de clôture

Ces primitives sont plus typiquement utilisées en fin de tâche. Pour sauvegarder une valeur (ou un vecteur) d'un *tableau3D* dans un agent, se déplacer ou tuer un agent.

SauverValeurs Renvoie un tableau 3D contenant les valeurs prises par un attribut spécifié mais commun aux agents de la liste d'entrée. La dimension du tableau dépend du nombre d'agents en entrée, du fait qu'il s'agit ou non d'historiques, et de la dimension de l'atribut lui-même (paramètre standard ou liste).
Argument d'entrée : un (ou une liste d') agent(s) ou historique(s) en entrée
Argument de sortie : un *Tableau3D* en sortie

DéplacementCiblé Trois modes de fonctionnement sont possibles (pour animats uniquement) :

- mettre l'animat directement sur la cellule d'entrée
- se rapprocher de la cellule cible selon une certaine vitesse (nb. de cellules / pas)

– s'éloigner de la cellule cible selon une certaine vitesse (nb. de cellules / pas).

Argument d'entrée : une cellule
Argument de sortie : la même cellule

Tuer Tue le ou les Animat(s) passé(s) en entrée.
Argument d'entrée : un animat ou une liste d'animats
Argument de sortie : aucun

3.4.6 Primitive de contrôle

Permet de sortir du mode strictement enchaîné des primitivesă: arrêter le flot d'exécution, l'exécuter en boucle ou dérouler la liste fournie en entrée pour donner les objets un par un aux primitives suivantes.

Conditions (Si ... Alors) Si les conditions sont remplies, alors on poursuit l'exécution des primitives de cette tache. Sinon, arrêt de l'exécution

TantQue / FinTantQue Boucle conditionnelle, elle doit se clore par *fin tant que*

DéroulerUneListe / FinDéroulerUneListe Départ d'une boucle qui doit se clore par *fin derouler une liste*. Cette primitive prend les éléments de la liste un par un et le fournit aux primitives suivantes. Elle se termine lorsque la liste est finie ou lorsque des conditions sont remplies.
Argument d'entrée : une liste d'objets
Argument de sortie : un élément de la liste

Chapitre 4

Définir un expérience simulatoire

Le **batch** permet d'enchaîner des simulations en faisant varier des paramètres et de sauver les résultats. Avant de démarrer des opérations batch de simulations il est important de :

- Toujours **sauver son projet**, car si celles-ci sont interrompues, la valeur des paramètres reste celle de l'experience interrompue, et non pas les valeurs initiales. En cas d'interruption, volontaire ou non, il faut recharger immédiatement le projet à partir du fichier de sauvegarde pour restaurer la valeur des paramètres.
- Donner la tâche *Batch* (voir la section 3.3.10 *Batch*) à un agent chargé de mémoriser les résultats : *une cellule* ou *un agent non situé*, pour lequel seront cochés les attributs à mémoriser et les dates de sauvegarde (défaut : en fin de simulation). Sinon, les simulations s'enchaîneront mais les résultats seront perdus.
- Donner un nom de fichier de sauvegarde des résultats. L'extension **.dat** sera automatiquement ajoutée. On aura en sortie un fichier texte qui peut s'ouvrir avec un tableur standard. Il est au format d'un fichier de type *HistoireDuMonde*, dans lequel l'attribut *date* (en jour) donne le numéro de l'experience, dans l'ordre où elles sont réalisées. Le tableau donne ensuite la valeur prise par les paramètres de chaque simulation et la valeur des variables de sorties mémorisées.
- Le champ *nb. max animats* donne le nombre maximum d'animats autorisé. Si ce chiffre est dépassé, la simulation en cours est arrêtée et la valeur *NaN (Not a Number)* est assignée aux variables de sortie. La procédure passe à la simulation suivante.
- Pour arrêter le batch, taper *Ctrl Y* puis *Terminate* (ou *proceed* pour reprendre) dans la fenêtre qui s'affiche. Comme déjà dit, immédiatement recharger son projet à partir de la dernière sauvegarde pour restaurer la valeur des paramètres.

Remarques importantes

- Une série d'expériences simulatoires mémorise le chemin d'accès aux paramètres. **Si par la suite les agents sont modifiés, certains accès peuvent devenir obsolètes** provoquant une erreur à l'exécution...
- Dans certains cas, si un fichier *.dat* est déjà ouvert par une autre application (Excel par exemple) le fait de relancer le batch provoque un plantage de Mobidyc par refus d'écraser ce fichier.
- Le mode Batch désactive automatiquement la sauvegarde des résultats. Mais dans le cas ou certaines tâches utilisent la mémoire des agents *(HistoriqueAgents)*, on peut alors forcer cette mémorisation via les préférences (menu Options $->$ Préférences $->$ Batch).

4.1 Exploration paramètre par paramètre

La figure 4.1 représente la boîte de dialogue qui sert à construire une exploration paramètre par paramètre. Cette procédure propose de calculer la réponse d'un modèle pour la gamme de valeurs spécifiées pour chaque paramètre.

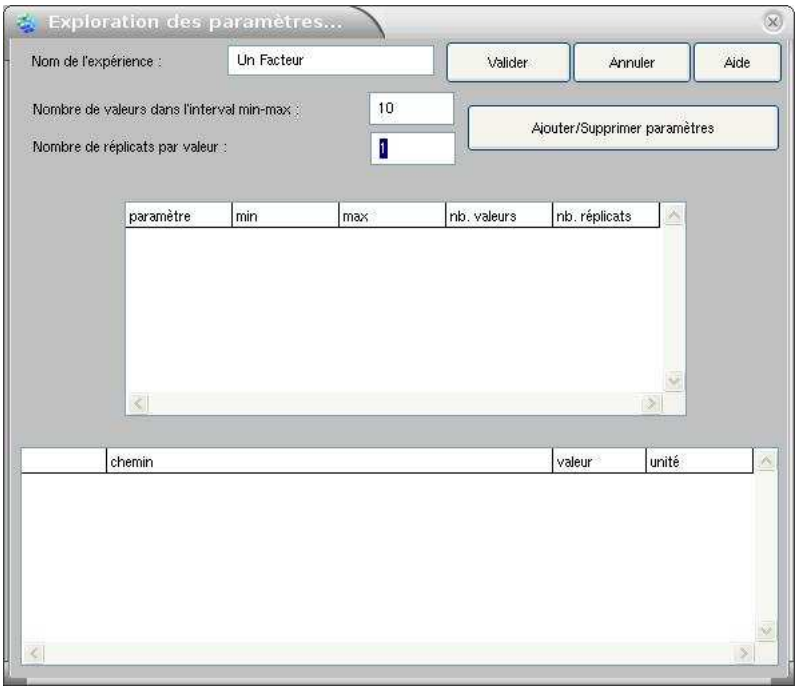

FIG. 4.1: Exploration paramètre par paramètre

Procédure

- Dans le champ de saisie *Expérience simulatoire* de la boîte de dialogue principale, utiliser le menu contextuel et sélectionner *Exploration des paramètres*.
- Saisir le nom de l'expérience simulatoire, le nombre de valeurs souhaitées (dans la pratique entre 10 et 30 pour avoir une bonne idée de la réponse du modèle au paramètre), et le nombre de réplicats. Il y aura donc $nbValeurs * nbRpliquats * nbParamtresSimulations$.
- Cliquer sur *ajouter/supprimer des paramètres*. les paramètres modifiables sont **en vert**. Les sélectionner pour ajouter leur chemin d'accès à la liste du bas. Pour supprimer une erreur, sélectionner le chemin à supprimer dans la liste du bas et utiliser le menu contextuel. Quitter la fenêtre en validant.
- Entrer la valeur mini et maxi des paramètres dans le tableau. *attention de bien respecter la correspondance colonne → numéro de parametre*. La boîte du bas rappelle cette correspondance. Il est possible de personnaliser le nombre de valeurs et le nombre de réplicats pour chaque paramètres.
- Valider. Avant de lancer le Batch, vérifier
  - qu'un agent non situé (ou une cellule) ait une tâche Batch en activité,
  - qu'un fichier de sauvegarde (non ouvert) soit spécifié,
  - que le projet soit sauvé.

4.2 Plan d'expérience complet

La procédure *plan complet* lance une simulation pour chaque combinaison possible de chaque valeur de chaque paramètre.

- En **procédure manuelle** entrer les niveaux à la main,
- en **procédure automatique** spécifier *val min*, *val max* et nombre de niveaux.

Attention, le nombre de combinaisons croit très vite. Pour 4 valeurs et 5 paramètres et 10 répliquats, on a $(4^5) * 10 = 10240$ *simulations*.

Typiquement, un plan complet est fait pour être dépouillé par une Analyse de Variance qui vous permettra de détecter notamment les interactions (synergies, antagonismes) entre paramètres. La figure 4.2 représente la boîte de dialogue de ce type d'experience simulatoire.

Procédure

- Dans le champ de saisie *Expérience simulatoire* de l'interface principale, utiliser le menu contextuel et sélectionner *Plan complet*.
- Saisir le nom de l'expérience, et le nombre de réplicats.

- Cliquer sur *ajouter/supprimer des paramètres*. Les paramètres modifiables sont en vert. Les sélectionner pour ajouter leur chemin d'accès à la liste du bas. Pour supprimer une erreur, sélectionner le chemin à supprimer dans la liste du bas et utiliser le menu contextuel. Quitter la fenêtre en validant.
- Entrer les valeurs (on parle de *niveaux* en Anova) souhaitées pour chaque paramètres. Dans la pratique, et en particulier lorsqu'on souhaite utiliser le module statistique que nous avons développé sous **R**, on utilise deux (si on suspecte un effet linéaire du paramètre) ou quatre niveaux (si on suspecte un effet quadratique). La procédure *Exploration des paramètres* permet justement d'aider à choisir entre ces deux alternatives, et à définir la valeur des différents niveaux. *Attention de bien respecter la correspondance colonne → numéro de parametre. La boîte du bas rappelle cette correspondance.*
- Avec la procédure automatique, entrer la valeur mini et la valeur maxi, puis le nombre de niveaux (2 niveaux minimum). Le premier niveau est toujours valeur min, le dernier valeur max, les niveaux sont équirépartis.
- Valider. Avant de lancer le Batch, vérifier qu'un agent non situé (ou une cellule) ait une tâche Batch en activité, qu'un fichier de sauvegarde

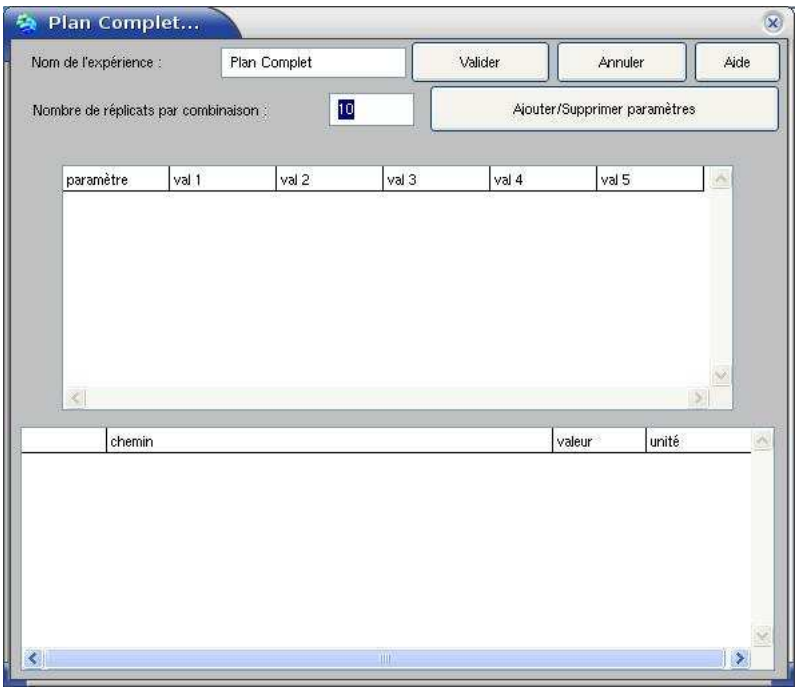

FIG. 4.2: Plan d'expérience complet

(non ouvert) soit spécifié, et que le projet soit sauvé.

4.3 Plan d'analyse de sensibilité

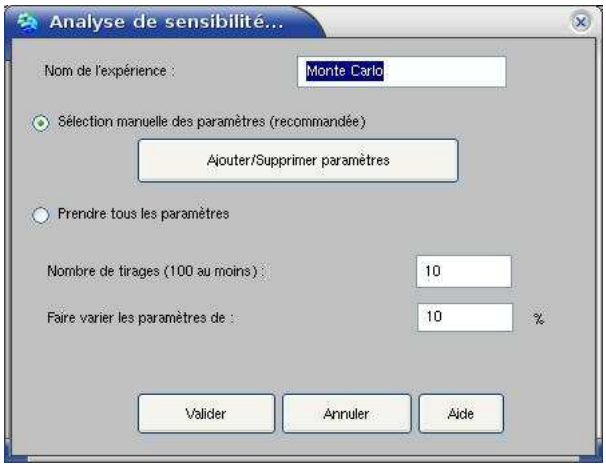

FIG. 4.3: Plan d'analyse de sensibilité

La procédure *Analyse de sensibilité* cherche à évaluer les *fonctions de sensibilité* de chaque paramètre, c'est à dire la dérivée de la réponse du modèle par rapport à chaque paramètre. Plus la sensibilité est forte, plus le modèle réagit à une petite fluctuation du paramètre.

Pour calculer cette sensibilité, on évalue la réponse du modèle au voisinage de la valeur nominale du paramètre, les autres paramètres gardant en principe une valeur fixe. Puis on calcule la pente de cette réponse par une régression linéaire. Il faut donc que cette réponse soit linéaire, ce qui implique de ne pas trop s'éloigner de la valeur nominale du paramètre. Une valeur de 10% convient souvent. Pour limiter le nombre de simulations, une variante consiste à faire varier *simultanement* tous les paramètres autour de leurs valeurs nominales.

Toutes les simulations peuvent alors être utilisées lors du calcul de la sensibilité de chaque paramètre. On obtient une mesure de sensibilité parfois légèrement différente, mais également intéressante dans la mesure où elle représente la sensibilité moyenne du paramètre, les autres étant également fluctuants autour de leur valeur nominale. C'est cette deuxième variante qui est proposée.

Cette mesure de sensibilité peut être répétée dans le temps, à différents pas de temps de la simulation. On obtient alors la courbe de l'évolution de la sensibilité du modèle au paramètre tout au long de la simulation, courbe toujours très informative. Le dépouillement de l'analyse de sensibilité

se termine typiquement par le calcul des corrélations entre ces différentes courbes. Si deux paramètres ont des courbes de sensibilité trop corrélées, cela signifie qu'ils ont le même effet sur le modèle tout au long de la simulation, et que le modèle est vraisemblablement sur-paramétré. La figure 4.3 représente la boîte de dialogue qui permet de définir ce type d'expérience simulatoire.

Procédure

- Dans le champ de saisie *Expérience simulatoire* de l'interface principale, utiliser le menu contextuel et sélectionner *Analyse de sensibilité locale*.
- Saisir le nom de l'expérience, le nombre de tirages (simulations) à effectuer (au moins 100) et la valeur de la fluctuation.
- Sélectionner *Tous les paramètres* ou cliquer sur *ajouter/supprimer des paramètres* pour restreindre l'analyse à certains paramètres. Les paramètres modifiables sont en vert. Les sélectionner pour ajouter leur chemin d'accès à la liste du bas. Pour supprimer une erreur, sélectionner le chemin à supprimer dans la liste du bas et utiliser le menu contextuel. Quitter la fenêtre en validant.
- Valider. Avant de lancer le Batch, vérifier qu'un agent non situé (ou une cellule) ait une tâche Batch en activité, qu'un fichier de sauvegarde (non ouvert) soit spécifié, et que le projet soit sauvé.

4.4 Hyper cube latin

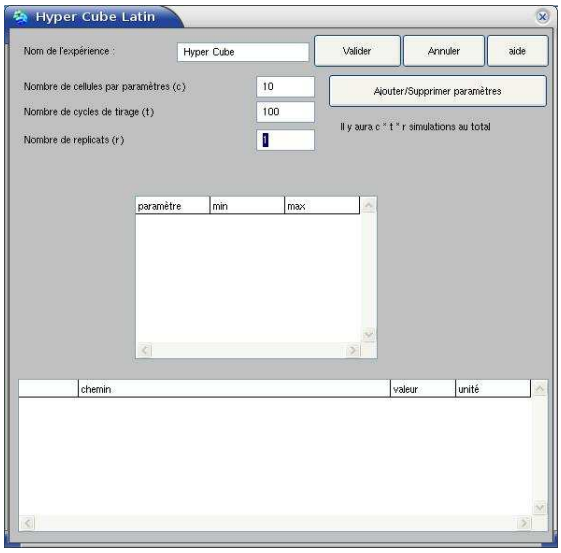

FIG. 4.4 – *Hyper cube latin*

Non commenté

4.5 Saisie manuelle du plan

Cette procédure permet de saisir manuellement les combinaisons des valeurs de paramètres qu'on souhaite tester (jusqu'à 5 paramètres différents). Les autres paramètres gardent leur valeur originelle, donc en générale celle du moule. Chaque combinaison peut être répliquée autant de fois que souhaité. La figure 4.5 représente la boîte de dialogue qui permet dedéfinir ce type d'expérience simulatoire.

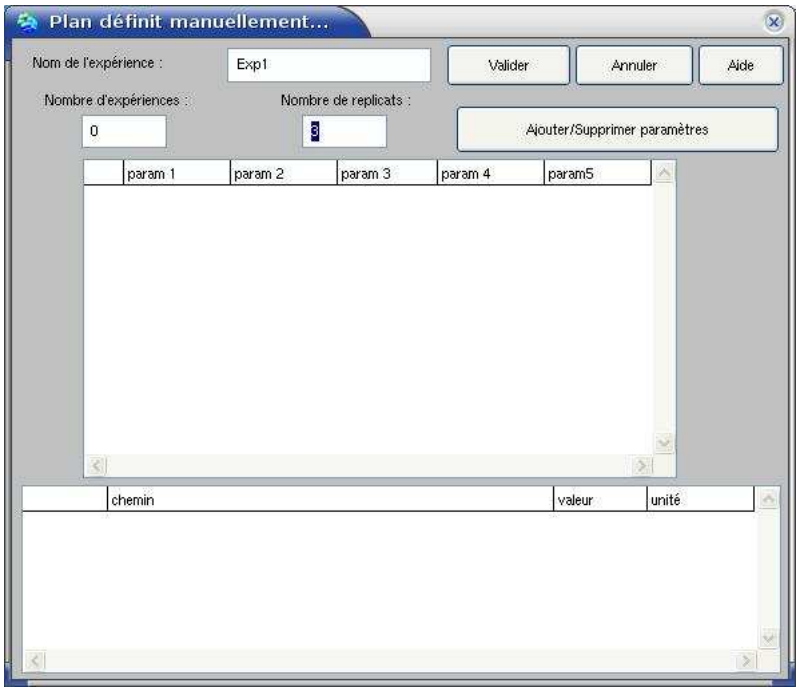

FIG. 4.5: Saisie manuelle du plan

Procédure

- Dans le champ de saisie *Expérience simulatoire* de l'interface principale, utiliser le menu contextuel et sélectionner *Saisie manuelle du plan.*
- Saisir le nom de l'expérience simulatoire, le nombre d'expériences, et le nombre de réplicats. Le nombre d'expériences est le nombre de combinaisons de valeurs de paramètres. (par défaut : 0, on garde les valeurs courantes des paramètres telles qu'elles se trouvent dans les moules des agents). Le nombre de réplicats est le nombre de répétitions à l'identique d'une expérience.
- Pour modifier des paramètres, entrer un nombre non nul d'expériences puis cliquer sur *ajouter/supprimer des paramètres.* La fenêtre qui s'af-

fiche permet de visualiser en vert les paramètres modifiables. Les sélectionner pour ajouter leur chemin d'accès à la liste du bas. Pour supprimer une erreur, sélectionner le chemin à supprimer dans la liste du bas et utiliser le menu contextuel. Il est possible de sélectionner jusqu'à cinq paramètres simultanément. Quitter la fenêtre en validant.

- Entrer la valeur des paramètres en colonnes dans le tableau. Attention de bien respecter la correspondance colonne → Numéro de parametre. La boîte du bas rappelle cette correspondance. Chaque ligne correspond à une expérience simulatoire. Bien ajuster le nombre d'expériences à la taille du tableau (au besoin modifier la valeur *nombre d'expériences*).
- Valider. Avant de lancer le Batch, vérifier qu'un agent non situé (ou une cellule) ait une tâche Batch en activité, qu'un fichier de sauvegarde (non ouvert) soit spécifié, et que le projet soit sauvé.

Troisième partie

Annexes

Chapitre 1

The GNU General Public License

Version 2, June 1991

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You

must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions For Copying, Distribution and Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.
   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.
   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.
3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sec-

tions 1 and 2 above on a medium customarily used for software interchange; or,

(b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

(c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.
8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

No Warranty

11. Because the program is licensed free of charge, there is no warranty for the program, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holders and/or other parties provide the program "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction.

12. In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the program to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.

End of Terms and Conditions

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> one line to give the program's name and a brief idea of what it does.
> Copyright (C) yyyy name of author
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) yyyy name of author
> Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
> This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w` and `show c`; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program
> 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> signature of Ty Coon, 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Chapitre 2

Installer Mobidyc

Mobidyc version 2.2, août 2005, lisez-moi Copyright INRA, tous droits réservés (licence GPL).

Remarque importante : Le nouveau module d'édition de formules mathématiques (MatScript) a été développé en Java. Il nécessite l'installation d'une machine virtuelle Java, version J2SE 1.4.1 ou plus récente de Sun Microsystems. A télécharger si nécessaire sur le site http://java.sun.com/j2se.

2.1 Installation et lancement de Mobidyc *Runtime*

Décompacter l'archive.

- **Mac** et **Windows :** double clic sur l'archive.
- **Unix/Linux :** unzip nomDeLarchive.

Pour obtenir l'utilitaire de décompactage :

`http://www.cdrom.com/pub/infozip/Zip.html).`

On obtient huit répertoires plus deux fichiers (celui-ci et un tutorial) :

- **Bin.** Contient le moteur VisualWorks Smalltalk
- **Image.** Contient les fichiers nécessaires à l'exécution de Mobidyc
- **Messages.** Contient l'aide en ligne et les traductions multi-langues du logiciel.
- **Projets.** Répertoire conseillé pour mettre vos propres modèles.
- **MB_exemples.** Contient les fichiers des modèles livrés à titre d'exemples en français.
- **MB_examples.** Contient les fichiers des modèles livrés à titre d'exemples en anglais.
- **MB_biespiele.** Contient les fichiers des modèles livrés à titre d'exemples en allemand.
- **MB_plugin.** Contient les mises à jour et modules en essai.

Puis :

- **Mac :** ouvrir le répertoire *image* puis double clic sur *mobidyc.im*
- **Windows :** ouvrir le répertoire *image* puis double clic sur *mobidyc.im*. Au premier lancement, associer les fichiers *.im* à l'éxécutable *visual.exe* qui se trouve dans le répertoire *bin*
- **Unix/Linux :** se placer dans *image* puis entrer la commande :

  ```
  ../bin/<system>/visual mobidyc.im
  ```

  ou mieux, modifier les règles du PATH. Il faut également spécifier la variable d'environnement VISUALWORKS. Pour cela (à vérifier), selon le cas :
  - VISUALWORKS=your/path/to/mobidyc ; export VISUALWORKS (dans le .bash_profile)
  - setenv VISUALWORKS 'your/path/to/mobidyc' (dans le .shrc sous Unix/Solaris).
- **Mac OS 9.** Il est conseillé d'augmenter la mémoire allouée à VisualWorks. Pour cela, localiser l'application *Visual* qui se trouve sous *bin:powermac*, et allouer autant de mémoire que vous pouvez (si possible 200 Mo, voire plus), dans l'onglet *mémoire* de ses propriétés.

L'installation est réussie si vous obtenez bien l'aide en ligne.

2.2 Utilisation des exemples

Commencez par *JeuDeLaVie* ou *Chevre-Loup*. Charger l'exemple (menu Programme $->$ chargerProjet). Si nécessaire, activer la visualisation (Outils $->$ Visualisation), et lancer une simulation. Promenez-vous ensuite dans les interfaces de visualisation des résultats et consultez l'aide en ligne.

Exemple *DeAngelis* (cannibalisme chez les poissons). Du fait des interactions, cet exemple est assez lent. Il est fourni avec deux tâches de chasse pour les animats : une tache assemblées à l'aide de primitive, et une tâche codée à l'aide de l'outil d'aide à la programmation. Vous pouvez désactiver la tâche assemblée et activer la tâche codée (menu peuplement $->$ définir les animats, et case à cocher dans le dictionnaire des tâches). L'exemple s'exécute environ 5 fois plus vite.

2.3 Installation de MathScript

Mathscript est la nouvelle interface qui permet de créer et modifier des fonctions mathématiques. Ce programme est écrit en java et les échanges entre la Mobidyc et Mathscript se font via des fichiers XML. L'appel du programme Mathscript est automatique chaque fois que l'utilisateur veut

exécuter la tâche éponyme, mais pour que Mathscript fonctionne il faut avoir au minimum *Java Runtime Environment* installé sur son ordinateur. Ceci est très fréquent car java est utilisé dans de nombreux domaines, de plus il peut faire parti des plugins installés pour le navigateur Web de la machine.

Dans le cas ou il est nécessaire d'installer le runtime Java[1] il suffit de se reporter à la page d'instruction de Sun :
http://java.sun.com/j2se/ paragraphe *How is it Delivred?*
Une fois la machine Java installée il suffit de vérifier que le fichier *MathEdit.jar* soit bien présent dans le répertoire *bin* de Mobidyc. L'appel de la tâche *mathscript* lancera automatiquement le programme Java, si rien ne ce passe cela signifie que l'installation du moteur Java est défectueuse ou (plus fréquement) que le PATH correspondant au programme ne fait pas partie des régles de recherche des programmes.

1. En fait il existe 2 possibilités :
   - soit installer uniquement le *runtime* qui est le moteur java permettant de faire tourner les applications Java
   - soit installer la disribution complète de Java qui permet également de développer des applications

Index

Bibliographie

[1] J. Ferber*Les systèmes Multi-agents vers une intelligence collective*, InterÉditions (1997)

[2] V. Ginot, C. Le Page, S. Souissi *A multi-agents to enhance end-user individual-based modelling* Ecological Modeling Vol. 157 (2002) 23-41

[3] C. Le Page, V. Ginot *Vers un simulateur générique de la dynamique des peuplements piscicoles* JFDIADSMA Intelligence Artificielle et Systèmes Multi-Agents (1997)

[4] V. Ginot, C. Le Page *Mobidic, a generic Multi-Agents Simulator for Modeling Populations Dynamics* 11th International Conferences on Industrial abd Engineering Applications of Artificial Intelligence and Experts Systems. Benicassim, Spain (1998)

[5] J. Ferber *Les sytèmes multi-agent. Vers une intelligence collective* Paris, Interéditions (1995)

[6] P. Hogewer, B. Hesper *Individual-oriented modelling in ecology,* Mathematical Computing Modelling 13 (6) 83-90 (1990).

[7] H. Lorek, M. Sonnenschein *Modelling and simulation software to support individual-based ecological modelling.* Ecological Modeling, 115 199-216 (1999)

[8] H. Richard *Formalisation des modèles de Mobidyc : Étude des choix techniques et mise au point d'un schéma XML.* Rapport de projet de fin d'étude, DESS TAII, Université d'Avignon (2003)

[9] A. Drogoul, J. Freber *Multi-agent Simulation as a Tool for Modelling Societes: Application to Social Differentitiation in Ant Colonies.* (1994)

[10] L. Magnin *Modélisation et simulation de l'environnement deans les systèmes multi-agents. Application aux robots footballeurs.* Thèse de doctorat, Université Paris VI

[11] D. Houssin *ATOLL : Un langage de description de modèle dédié à la dynamique des peuplements.* Rapport de DEA, Université Paris VI (1998)

[12] D. Houssin, S. Bornhofen, S. Souissi, V. Ginot *Entre programmation par composants et langages d'experts.* Techniques et Sciences Informatiques, RSTI - TSI - 21/2002 p. 525 à 548

[13] B. McLaughlin *Java et XML* 2^{ieme} *édition* Éditions O'Reilly 2002

[14] E. van der Vlist *XML schéma* Éditions O'Reilly 2002

[15] J-J. Thomasson *Schémas XML* Éditions Eyrolles 2003

[16] L. Dirat *Outils pour des Mathématiques Interactives et Distribuées* thèse de doctorat, sciences et technologies de l'information et de la communication, soutenue le 2 janvier 2001, Université de Nice - Sophia Antipolis.

[17] D. Riehle, M. Tilman R. Johnson *Dynamic Object Model.* Conference on Pattern Languages of Programs 2000 (PLoP 2000). Washington University, Technical Report number : wucs-00-29.