Exploration sémantique des modèles socio-environnementaux

Hasina Lalaina Rakotonirainy

THÈSE

En vue de l'obtention du grade de

DOCTEUR DE L'UNIVERSITE DE FIANARANTSOA

ECOLE DOCTORALE MODELISATION – INFORMATIQUE

Equipe d'Accueil : **Modélisation des Systèmes Complexes**

Domaine : **Science et Technologie**

Spécialité : **Informatique**

Intitulée

Exploration sémantique des modèles socio-environnementaux

-

Approche générique pour l'initialisation et l'observation des modèles de simulation complexes

présentée le 18 Mai 2016

par **RAKOTONIRAINY Hasina Lalaina**

Membres de Jury

MM.

Président	RAZAFIMANDIMBY Josvah Paul	Professeur Titulaire, Université de Fianarantsoa
Co-Directeurs	RAMAMONJISOA Bertin Olivier	Professeur Titulaire, Université de Fianarantsoa
	MÜLLER Jean-Pierre	Senior Researcher, CIRAD-ES-GREEN, Montpellier
Rapporteurs	TRAORE Mamadou Kaba	Professeur, Université Blaise Pascal, Clermont-Ferrand 2
	BRUEL Jean-Michel	Professeur des Universités, Université de Toulouse
Examinateurs	RAKOTONDRAOMPIANA Solofo	Professeur Titulaire, Université d'Antananarivo
	MAHATODY Thomas	Maître de Conférences, Université de Fianarantsoa
Invités	HERVE Dominique	Chercheur, IRD - GRED, Montpellier
	HAJALALAINA Aimé Richard	Maître de Conférences, Université de Fianarantsoa

Il n'y a ni sagesse, ni intelligence, ni conseil, en face de l'Eternel.
Enseigne-moi, Eternel, la voie de tes statuts,
pour que je la retienne jusqu'à la fin!
Donne-moi l'intelligence, pour que je garde ta loi
et que je l'observe de tout mon cœur.

Psaumes 119:33-34, Proverbes 21:30

A Lui soit la gloire aux siècles des siècles; Amen!

Remerciements

Je tiens particulièrement à nommer, remercier et exprimer ma profonde reconnaissance aux personnes qui m' ont prêté assistance, soutenu et aidé, m'ayant ainsi permis d'atteindre aujourd'hui ce niveau.

- Monsieur RAMAMONJISOA Bertin, Directeur de l'EDMI (Ecole Doctorale Modélisation - Informatique), Université de Fianarantsoa, Co-Directeur de Thèse. Malgré la distance, il a toujours été attentionné et vigilant sur mon parcours et n'a cessé de m'encourager et de me conseiller pour le bon déroulement de mes recherches et mon insertion professionnelle. Sa passion pour le développement durable m'a impressionné et m'a incité à orienter mes recherches pour contribuer au même développement.
- Monsieur MULLER Jean-Pierre, Cadre Scientifique, CIRAD-ES-GREEN, Co-Directeur de thèse. Il a été mon professeur à l'ENI (Ecole Nationale d'Informatique), Université de Fianarantsoa depuis 2009, pendant mes études approfondies (DEA) et m'accompagne jusqu'à ce jour. Il m'a initié à la recherche et m'a amené à aimer la profession de chercheur. La confiance et la disponibilité dont il a fait preuve à mon égard pendant mon parcours à ses côtés m'ont aidé à aller de l'avant et ont contribué à mon intégration dans le métier.

  Il a été comme un père pour moi à l'étranger et m'a toujours réconforté et conseillé dans plusieurs domaines de ma vie. Je ne manquerai pas de le citer comme modèle à mes étudiants.
- Monsieur RAZAFIMANDIMBY Josvah Paul, Enseignant-Chercheur à l'ENI, Université de Fianarantsoa, qui nous fait l'honneur de présider cette soutenance. Il a également voulu m'accompagner et m'encourager dans mon cheminement à travers les années jusqu'à aujourd'hui et ce, parfois malgré la distance.
- Monsieur TRAORE Mamadou Kaba, Professeur à l'Université Blaise Pascal, Clermont-Ferrant 2, pour sa bienveillance et sa disponibilité d'avoir bien voulu être mon Rapporteur dans le présent travail.
- Monsieur BRUEL Jean Michel, Professeur des Universités, IRIT (Institut de Recherche en Informatique de Toulouse), Université de Toulouse, également pour sa bienveillance et sa disponibilité d'avoir bien voulu être mon Rapporteur dans le présent ouvrage.

- Monsieur RAKOTONDRAOMPIANA Solofo, Enseignant-Chercheur, IOGA (Institut et Observatoire de Géophysique d'Antananarivo), Université d'Antananarivo pour sa bienveillance et sa disponibilité d'avoir bien voulu accorder de son précieux temps pour apprécier et examiner le présent ouvrage.
- Monsieur MAHATODY Thomas, Enseignant-Chercheur à l'ENI, Université de Fianarantsoa pour sa bienveillance d'avoir également bien voulu accorder de son précieux temps pour apprécier et examiner le présent ouvrage.
- Monsieur HAJALALAINA Aimé Richard, Directeur du CUFP (Centre Universitaire de Formation Professionnalisante), Université de Fianarantsoa, collègue précieux qui m'a toujours soutenu et a bien voulu accorder de son temps pour être présent durant ce moment.
- Les différentes personnalités de l'Université de Fianarantsoa, notamment Monsieur RAFAMATANANTSOA Fontaine, Président de l'Université, Messieurs RATOLOJANAHARY Faniry, RANIRIHARINOSY Karyl, RAZAFINDRAKOTO Raft, RAZANAKA Samuel, RANDRIAMAHEFARISON Jean Rémi, DINAHARISON Yves José, Membres du Conseil Scientifique de l'EDMI. Ils ont tous pris part et veillé au bon déroulement de mes recherches.
- Madame AUBERT Sigrid, Messieurs HERVE Dominique et HARIFIDY Rakoto Ratsimba, Membres du Comité de Thèse, qui ont suivi de près l'évolution de mon travail durant ces trois années et m'ont prodigué de bons conseils à ce propos.
- Mesdames BOTTA Aurélie, ANTONA Martine, ROVIS Nathalie, PIKETTY Marie-Gabrielle et FALLOT Abigaïl, Messieurs BOUSQUET François, TOURRAND Jean-François, BAZILE Didier, BOMMEL Pierre, LE PAGE Christophe, QUESTE Jérôme et encore bien d'autres, dans l'Equipe GREEN du CIRAD qui a été mon équipe d'accueil à Montpellier durant trois années. Ils sont intervenus, chacun d'une manière spéciale et à sa façon dans mon parcours, tant affectif que professionnel. Cela m'a beaucoup touché et a contribué à mon intégration au sein de l'Equipe.
- L'AIRD (Agence Inter-établissements de Recherche pour le Développement) qui m'a accordé une bourse de financement durant ces trois années, par le biais de son programme d'appui au renforcement et à la consolidation des potentiels des chercheurs dans les pays du Sud.
- L'Equipe GDHD (Gouvernance-Développement Humain Durable) à Madagascar qui a contribué à enrichir mes expériences par les séminaires, les formations, les descentes

sur terrain et les partages de compétences sur l'interdisciplinarité avec les autres chercheurs des Universités malagasy.

- Le CIRAD et l'Equipe du DP "Forêts et Biodiversité" à Madagascar. Les réunions avec les thématiciens à l'initiative de l'Equipe ont été enrichissantes pour l'avancement et l'amélioration de mes travaux.
- Les personnes que j'avais l'opportunité de rencontrer lors des diverses conférences sur l'IDM (Ingénierie Dirigée par les Modèles) à Paris, Bordeaux, Lyon, Toulouse et Montpellier. Elles ont témoigné de l'intérêt pour mon travail, ce qui a également contribué à l'avancement de mes travaux par leurs conseils et leurs partages bénéfiques.
- Mes collègues doctorants, Paulo, Diallo, Elena, Gabriel, Anissou, Gabin, Anthonio, Hermine, Pauline, Ola, Francesca, Qi, Sitraka, Thierry, Alexio, Manitra, Angela, Camille, Arthur, Amaury et Caroline. Je ne saurais oublier notre soutien et nos encouragements mutuels dans toutes les situations diverses que nous avons traversées ensemble.
- La famille MICLET Gérard qui m'a accueilli à bras ouverts à Clapiers et m'a témoigné son soutien moral. Leur hospitalité et leur bienveillance m'ont beaucoup touché.
- L'Eglise FPMA et les compatriotes à Montpellier qui m'ont soutenu et intercédé dans la prière pour moi. Ils ont constitué une véritable famille pour moi à l'Etranger. « Tsy misy mivatsy havana tokoa fa hianareo no havana novantanina sy nanohana tamin'ny fotoana rehetra ».
- Mon père qui, même de loin, n'a cessé de me soutenir en prières et en conseils. Actuellement, nous sommes collègues et je suis fier d'être son fils, d'avoir suivi ses pas dans le métier.
- Ma mère, avocate et membre principal de mon comité de lecture, passionnée par mes travaux au fil des années, ne manquait jamais de lire tous mes documents et ouvrages, tout en veillant sur la forme et sur le fond, à tel point qu'à force de s'en imprégner, elle est devenue spécialiste en la matière. J'estime qu'elle mérite une médaille.
- Mes frères et sœur qui m'ont soutenu dans la prière.
- Mes beaux-parents, mon beau frère, ma belle sœur et leur famille respective qui n'ont cessé de prier pour moi.

- Ma chère moitié, Christelle, qui a toujours été présente et affectueuse, que ce soit dans les bons moments ou les situations difficiles.
- Enfin, tous ceux qui, de près ou de loin, ont contribué à l'élaboration et la réalisation de cette thèse.

Merci infiniment et que Dieu vous bénisse tous abondamment.

Résumé

Les chercheurs veulent aborder toute la complexité des socio-écosystèmes (SES) afférents aux dynamiques biophysiques, sociales ainsi qu'à leurs interactions. Afin d'aborder cette complexité, ils ont recours à des modèles de simulation de plus en plus complexes, dont l'initialisation et l'observation sont devenues difficiles à mettre en œuvre. Toutefois, aucun cadre générique n'a encore été développé pour résoudre ce problème. L'objectif de cette thèse est de proposer un cadre générique pour la spécification et la mise en œuvre de l'initialisation, à partir de nombreuses données hétérogènes, et l'observation pour produire les indicateurs souhaités par les thématiciens. Le résultat est un ensemble d'outils et de savoir-faire, permettant aux thématiciens de spécifier et d'automatiser l'ensemble du processus d'exploitation d'un modèle de simulation, de l'initialisation à la production des indicateurs. Pour cela, nous proposons de formuler l'initialisation et l'observation des modèles de simulation en des transformations entre données et structures de données. Cette formulation permet d'utiliser les concepts de l'ingénierie dirigée par les modèles (IDM) afin de mettre en œuvre des langages dédiés (DSL). Ces derniers fournissent les concepts nécessaires permettant aux thématiciens de spécifier plus facilement l'initialisation et l'observation de modèles de SES.

Mots-clés : Ingénierie Dirigée par les Modèles (IDM), Langage dédié, Initialisation, Observation, Modèle de simulation complexe, Socio-écosystème

Abstract

Researchers have sought to deal with the complexity of socio-ecosystems including biophysical and social dynamics, and their interactions. In order to cope with this complexity, they need increasingly complex models, whose initialization, and observation are becoming very difficult to implement. However, no generic framework has yet been developed to address this issue. The objective of the thesis is a generic framework for specifying and implementing the initialization from numerous heterogeneous data, and the observation producing the desired indicators. The result is a set of tools and know-how, allowing thematicians to specify and automate the whole process of exploitation of a simulation model, from the initialization to the production of indicators. For this, we propose to formulate the initialization and observation as transformations among data and data structures. This formulation allows to use the Model Driven Engineering (MDE) concepts in order to implement the generic framework and the corresponding domain specific languages (DSL) allow thematicians to specify easier initialization and observation SES models.

Keywords: Model Driven Engineering (MDE), Domain Specific Language (DSL), Initialization, Observation, Complex simulation model, Socio-ecosystem

Liste des figures

Table des codes

Liste des tableaux

Acronymes

ABM	Agent-Based Modeling
ACM	Analyse des Correspondances Multiples
AML	AtlanMod Matching Language
AMMA	ATLAS Model Management Architecture
AMW	ATLAS Model Weaver
AS	Abstract Syntax (Syntaxe Abstraite)
ATL	ATLAS INRIA & LINA
CASE	Computer Aided Software Engineering
CaTMAS	Carbon of Territory Multi-Agent Simulator
CIM	Computation Independent Model
CIRAD	Centre de coopération Internationale en Recherche Agronomique pour le Développement
CLASYCO	Conception de Langages dédiés au domaine des SYstèmes Complexes
ComMod	Companion Modelling
CS	Concret Syntax (Syntaxe Concrète)
CSV	Comma Separated Values
CUFP	Centre Universitaire de Formation Professionnalisante
DEVS	Discret Event Simulation Specification
DOM	Document Object Model
DSL	Domain Specific Language
DSMS	DataStream Management System
EBNF	Extended Backus-Naur Form
ECEC	Early Childhood Education and Care
EDMI	Ecole Doctorale Modélisation - Informatique
EDMMAS	Energy Demand Management by Multi-Agent Simulation
EMF	Eclipse Modeling Framework
EMOF	Essential MOF
EMP	Eclipse Modeling Project
ESR	Économie et Sociologie Rurales
ETL	Extract - Transform - Load
GAMA	GIS & Agent-based Modeling Architecture
GAML	GAMA Modeling Language
GBIF	Global Biodiversity Information Facility
GCF	Gestion Contractuelle des Forêts
GDHD	Gouvernance - Développement Humain Durable
GEF	Graphical Editing Framework
GELOSE	Gestion Locale des Ressources Naturelles Renouvelables
GIF	Graphics Interchange Format
GIMS	Graphical Interface for Modeling and Simulation
GIS	Geographic information system
GME	Generic Modeling Environment
GMF	Graphical Modeling Framework
GUI	Graphical User Interface
HTML	HyperText Markup Language

IAMM	Institut Agronomique Méditerranéen de Montpellier
IDE	Integrated development environment
IDM	Ingénierie Dirigée par les Modèles
IMAS	Impact des Modalités d'Accès aux Semences
INRA	Institut National de la Recherche Agronomique
INRIA	Institut National de Recherche en Informatique et en Automatique
IOGA	Institut & Observatoire de Géophysique d'Antananarivo
IPF	Iterative Proportional Fitting
IRISA	Institut de Recherche en Informatique et Systèmes Aléatoires
IRIT	Institut de Recherche en Informatique de Toulouse
JPEG	Joint Photographic Experts Group
LINA	Laboratoire Informatique de Nantes Atlantique
M&S	Modélisation et Simulation
M2M	Model To Model
M2T	Model To Text
MAELIA	Multi-Agents for EnvironmentaL norms Impact Assessment
MASC	Map Sectors Creator
MDA	Model Driven Architecture
MDE	Model Driven Engineering
MIMOSA	Méthode Informatique de Modélisation et Simulation d'Agents
MIRO	Modélisation Intra-urbaine des Rythmes quOtidiens
MOF	Meta-Object Facility
MVC	Modèle - Vue - Contrôleur
NSGA II	Elitist Non-Dominated Sorting Genetic Algorithm
OCL	Object Constraint Language
OMG	Object Management Group
OQL	Object Query Language
OWL	Web Ontology Language
PHP	Hypertext Preprocessor
PIM	Platform Independent Model
PNG	Portable Network Graphics
PoV	Point Of View (Point de Vue)
PSM	Platform Specific Model
QVT	Query - View - Transformation
QVTO	Query - View - Transformation Object
RNR	Ressource Naturelle Renouvelable
ROO	Représentation Orientée Objet
SD	Semantic Domain
SES	Socio-Ecosystème
SIC	Système d'Informations Complexes
SIEGMA	Stakeholders Interactions in Environmental Governance by Multi-Agent System
SIG	Système d'Information Géographique
SMA	Système Multi-Agents
SQL	Structured Query Language
TerraME	TerraLib Modeling Environment
TIFF	Tagged Image File Format

TRANSIMS	Transportation Analysis SIMulation System
UML	Unified Modelling Language
UNICEF	United Nations Children's Emergency Fund
UPR-GREEN	Unité Propre de Recherche Gestion des ressources Renouvelables
VLE	Virtual Laboratory Environment
VOI	Vondron'Olona Ifotony (Communauté locale de base)
VSML	Visual Semantic Unified Modeling Language
XELOC	eXtensible Editing Language Of Configuration
XML	Extensible Markup Language

Thèse préparée au sein de l'unité de recherche

GREEN

Gestion des Ressources Renouvelables et Environnement

Campus International de Baillarguet

34398 Montpellier cedex 5 - France

http://ur-green.cirad.fr/

Sommaire

Chapitre I. Introduction générale

Depuis une vingtaine d'années, les chercheurs s'intéressent au fonctionnement des socio-écosystèmes (SES) [Redman et al. 2004] qui sont faits d'un ensemble intégré et complexe d'écosystèmes et de sociétés humaines en interactions. Notamment, les thématiciens, i.e. les experts du domaine, considèrent que la gestion rationnelle des ressources naturelles renouvelables (RNR) peut devenir le pilier du développement humain durable [UNICEF 2010]. Plus précisément, ils estiment que la maîtrise des RNR permet, entre autres, de préserver la viabilité floristique et faunistique, d'améliorer le rendement, la production ainsi que les conditions de vie des foyers dans les communes et les régions.

Cependant, évaluer les impacts des décisions des parties prenantes de la gestion des RNR sur un SES donné n'est pas une chose facile. Les scientifiques ont mis en évidence l'enjeu de la compréhension partagée des interactions entre dynamiques sociales et dynamiques biophysiques caractérisant les SES.

Pour comprendre toute la complexité des SES afférents aux dynamiques biophysiques, sociales ainsi qu'à leurs interactions, de nombreux outils de modélisation (informatiques, mathématiques, statistiques, multi-agents, jeux de rôle, etc.) ont été développés. Les modèles engendrés sont ensuite utilisés comme des outils d'aide à la représentation, à la réflexion, à la décision et à la concertation. Ils deviennent le support du dialogue interdisciplinaire [Etienne 2010], [Hervé et Rivière 2015]. Le but des modèles de SES est en particulier de faciliter l'apprentissage mutuel aussi bien des acteurs que des parties prenantes (scientifiques, gestionnaires, propriétaires et utilisateurs) impliqués directement ou indirectement dans la gestion locale des RNR. Plus d'une trentaine de modèles sont mentionnés sur un site spécialisé [ComMod 2015]; citons quelques-uns à titre d'illustration :

- « AguaAloca » pour la gestion de l'eau dans un bassin versant périurbain [Clavel et al. 2008] ;
- « Camargue » pour les multi-usages et la biodiversité des marais à roseaux [Mathevet et al. 2008];
- « Domino» pour la médiation et la prospective territoriale à La Réunion [Daré et al. 2007] ;
- « Don Hoi Lord » pour la gestion de la pêcherie de couteaux de Don Hoi Lord [Worrapimphong 2005] ;

- « Frêne » pour le boisement spontané des montagnes de Bigorre [Monteil et al. 2008] ;
- « Kat Aware » pour les allocations multi-usages de l'eau en Afrique du Sud [Farolfi et al. 2008] ;
- « Lam Dome Yai » pour la gestion de la riziculture et les migrations au Nord-Est de la Thaïlande [Naivinit et al. 2010] ;
- « Larq'asninchej » pour l'urbanisation et les canaux d'irrigation en Bolivie [Vega et al. 2006] ;
- « Larzac » pour les dynamiques forestières sur le causse[1] du Larzac [Simon et Etienne 2010] ;
- « Lingmuteychu » pour les conflits de l'eau agricole au Bhoutan [Gurung et al. 2006] ;
- « Luberon » pour l'élevage et la dynamique des territoires en Luberon [Napoleone et al. 2008] ;
- « Méjan » pour l'envahissement du causse Méjan par les pins [Etienne et al. 2004] ;

Et aussi de nombreux autres existent dans la littérature, comme par exemple :

- « SIEGMA » [Gaudieux et al. 2014], un modèle pour étudier les comportements des fermiers et des forestiers dans la région d'Analamanga, District de Miarinarivo à Madagascar par rapport aux différentes politiques et lois sur la gestion des ressources naturelles ;
- « DS » [David et al. 2007], un modèle de simulation de la gestion des espaces fonciers de l'île de La Réunion qui tient compte des dynamiques de la population ainsi que de l'évolution de l'espace ;
- « EDMMAS », une évolution du modèle DS [Gangat et al. 2009] permettant d'aider les experts à la prise de décision avec la possibilité d'observer et d'exploiter différents scénarios de l'usage du sol et la planification de sa conservation sur l'ensemble de l'île de La Réunion ;
- « IMAS » [Belem et al. 2011a], un modèle pour tester des scénarios sur l'évolution de la diversité variétale au Chili et au Mali ;
- « CaTMAS » [Belem et al. 2011b], un modèle multi-agents pour simuler la dynamique des ressources de carbone dans les villages en Afrique de l'Ouest ;
- « Mirana » [Aubert et al. 2010], un modèle permettant d'évaluer l'impact du transfert de gestion des RNR aux communautés locales de base à Madagascar.

[1] Un plateau karstique fortement érodé caractéristique des auréoles sédimentaires du sud et de l'ouest du Massif central français

A chaque fois, l'utilisation de ces divers modèles pose deux questions principales :

- La question de l'initialisation de modèles sur la base d'un ensemble de scénarii pour les analyses prospectives en lien avec les acteurs locaux, ou d'un ensemble de valeurs de certains paramètres pour les analyses de sensibilité et pour les études plus scientifiques. A l'usage, la description de l'état initial à partir duquel on commence pour les simulations est tout aussi importante que le modèle lui-même et demande beaucoup de données.
- La question de l'observation de modèles de simulation et donc de l'extraction des indicateurs pertinents relatifs aux questions de recherche et/ou de développement posées par les thématiciens. Etant donné que les indicateurs sont des indices ou un résumé d'informations très utilisés par les thématiciens pour pouvoir mesurer l'évolution de différents éléments qui les intéressent dans un SES à des fins d'évaluation et/ou d'aide à la prise de décision ; de même, observer ce qui se passe pendant la simulation est aussi important que la simulation elle-même. L'enjeu est d'une part de comprendre et faire comprendre pourquoi la simulation produit certains résultats, et d'autre part d'évaluer les scénarii par rapport à des objectifs de gestion.

Face à ces questions, l'exploration des modèles de SES devient difficile, et demande un effort considérable de la part des informaticiens et des modélisateurs. Initialiser et observer un modèle de simulation nécessitent beaucoup de codage, d'une part afin d'extraire les données recueillies et de les faire correspondre aux modèles de simulation ; d'autre part, de satisfaire les besoins des thématiciens en termes d'observation de modèles de SES.

Pour répondre à ces questions, des chercheurs ont essayé d'élaborer des outils et/ou des démarches contribuant à faciliter le processus d'initialisation et d'observation de modèles en général. Cependant, les approches présentées sont relativement ad hoc et il n'existe pas encore de cadre général envisagé pour la problématique d'initialisation et d'observation du modèle.

Nous proposons donc, dans le cadre de cette étude, une approche générique en mettant à disposition des thématiciens, des outils et des savoir-faire, leur permettant de spécifier et d'automatiser l'ensemble du processus d'exploitation d'un modèle de simulation, de l'initialisation à la production des indicateurs. Cette approche se définit par la reformulation des questions d'initialisation et d'observation de modèle en un problème de transformation entre données et structures de données. Etant donné que nous avons affaire à des transformations de données et de structures de données, nous proposons de formuler la problématique qui précède en terme

d'ingénierie dirigée par les modèles (ou IDM). L'utilisation d'IDM nous permet de mettre en œuvre des langages dédiés (ou DSL) accessibles aux thématiciens.

Par conséquent, cette approche nous permettra de développer la dimension sémantique des modèles de simulation, à savoir la possibilité de décrire, de façon plus proche du discours des thématiciens et des acteurs locaux :

- Le contenu des sources de données indépendamment de leur support ;
- De quoi parle le modèle indépendamment des structures de données sous-jacentes;
- La structure des observables en termes d'indicateurs sur des données identifiables (revenus, diversité floristique et agronomique, coût du contrôle, etc.).

Puis de générer à partir de ces descriptions, le code informatique permettant d'initialiser et d'observer le modèle de simulation.

Ce travail de recherche vise donc à :

- Etendre les efforts des chercheurs, notamment, les informaticiens et les modélisateurs, en développant la dimension sémantique des modèles de simulation.
- Définir une méthode sémantique de spécification et de mise en œuvre de scénarii de simulation et des indicateurs que l'on souhaite suivre.
- Prendre en compte dans la conception du modèle de simulation, des données hétérogènes sur les aspects social (comportement de la population vis-à-vis de l'environnement : viabilité du système), économique (exploitation rationnelle de l'environnement : viabilité du système) et environnemental (durabilité du système).
- Mettre à disposition d'un outil informatique en vue d'une aide à la prise de décision et tenant compte du développement humain durable.
- Aider les thématiciens et les décideurs à la compréhension des dynamiques socio-environnementales.
- Réduire la complexité de la modélisation multi-niveau des SES en explorant un certain nombre d'outils et de technologies dédiés à ce genre de problème, en les mettant en œuvre et en montrant les conséquences.

Afin de mieux cerner la problématique sur les questions d'initialisation et d'observation de modèles de simulation, nous allons développer, au chapitre II, comment les modèles de simulation sont construits, de quelles données ils ont besoin pour fonctionner et qu'est- ce qui peut intéresser les thématiciens lors de la simulation. Nous choisissons un modèle de SES dénommé « Mirana » [Aubert et al. 2010] pour illustrer ces besoins.

Etant donné qu'il n'existe pas encore de cadre générique pour le problème d'initialisation et d'observation de modèles, au chapitre III, nous allons présenter quelques travaux qui ont été menés par les chercheurs autour de cette problématique pour repérer les éléments nécessaires à son élaboration afin d'en proposer un processus générique.

Le chapitre IV introduit et justifie le choix de l'utilisation d'IDM, ainsi que la mise en œuvre d'un certain nombre de langages dédiés (DSL) pour initialiser et observer les modèles de SES de manière générique.

Nous présentons respectivement, aux chapitres V et VI, le résultat de notre proposition, qui est la mise en œuvre et l'utilisation des DSL permettant de résoudre les problèmes d'initialisation et d'observation de modèles de SES. En se servant comme « fil rouge » d'un modèle de simulation très simple dénommé ECEC [Pepper et Smuts 2000], nous présentons les processus complets d'initialisation et d'observation de modèles que nous proposons.

Chapitre II. Problématique

2.1. Introduction

Afin d'établir la problématique sur les questions d'initialisation et d'observation des modèles de simulation, nous allons décrire comment sont construits les modèles de SES, de quelles données ils ont besoin pour pouvoir être initialisés et qu'est-ce qui peut intéresser les thématiciens pendant leur simulation ? Comme illustration, nous allons décrire dans ce chapitre, le modèle de simulation « Mirana » [Aubert et al. 2010], un modèle que nous développons au sein de l'équipe GREEN du CIRAD.

2.2. Modèle de simulation « Mirana »

2.2.1. Contexte

A Madagascar, l'Etat malgache s'est rendu compte que la surface des forêts de Madagascar a diminué d'une façon assez rapide en l'espace d'une soixantaine d'années. Si elle comptait plus de 53 millions d'hectares en 1927, il n'en restait plus que 12 millions en 1995 [Aubert et al. 2010]. Ce qui engendre une importante dégradation de l'environnement. Cette situation alarmante a amené les responsables étatiques à chercher des solutions pour une gestion cohérente, sécurisante et durable des RNR. Entre autres, des études ont été réalisées sur le *tavy*[2] qui est considéré comme principale cause de déforestation mais aussi comme moyen de production à Madagascar [Aubert et al. 2003]. Les différentes réflexions ont conduit à une élaboration d'un programme de gestion communautaire des RNR qui a abouti à l'adoption de deux types de contrats de conservation pour la gestion durable et pour la sécurité des ressources naturelles, à savoir la loi n° 96-0251 du 30 Septembre 1996 relative à la Gestion Locale des Ressources Naturelles Renouvelables (GELOSE) et le décret relatif à la Gestion Contractuelle des Forêts (GCF) en 2001 [Beuret 2010]. Des recherches, généralement, sur l'impact des transferts de gestion en faveur des communautés locales de base, ont été ensuite menées à travers différents projets depuis 1997 [Ramamonjisoa et al. 2012] afin d'évaluer les impacts de la mise en place de cette politique sur le développement durable. Parmi les différents projets, le modèle Mirana [Aubert et al. 2010] permettant d'évaluer l'impact du transfert de gestion des RNR aux communautés locales de base, (ou VOI pour Vondron'Olona Ifotony) a été construit.

[2]Culture de riz pluvial sans labour sur défriche-brûlis de forêt naturelle humide.
[3]http://mimosa.sourceforge.net/

2.2.2. Mirana : Un modèle de simulation

Mirana est un modèle de simulation permettant d'explorer et de discuter des scénarii de gestion intégrée des RNR [Aubert et al. 2010]. Il a été implémenté sur la plateforme informatique MIMOSA[3] (Méthode Informatique de Modélisation et Simulation d'Agents) développée par Müller J.-P. au sein de l'UPR GREEN du CIRAD [Müller 2004]. Inspiré de la notion d'institution de l'économiste Ostrom Elinor [Ostrom 2009] qu'elle définit comme étant « *des ensembles de règles opérationnelles utilisées pour déterminer ce qui est éligible pour prendre des décisions dans une certaine arène, quelles actions sont permises et prohibées, quelles règles d'action seront utilisées, quelles procédures seront suivies, quelle information doit ou ne doit pas être fournie et quels gains seront attribués aux individus en fonction de leurs activités* », le modèle Mirana sert de support à une recherche pluridisciplinaire sur la forêt d'Ambohilero, d'une superficie de 117 600 hectares dans la Commune Rurale de Didy située à 45 km d'Ambatondrazaka (Nord-Est de Madagascar) et dont la superficie totale est de 135 000 hectares et une population d'environ 21 000 habitants. Le but du modèle Mirana est surtout d'accompagner les modélisateurs et les différents acteurs du SES dans la démarche de modélisation d'accompagnement (ou ComMod pour Companion Modelling) [Aubert et al. 2010], [Barreteau et al. 2003] afin de mieux apprécier les impacts (les risques et les opportunités) du transfert de gestion des ressources forestières par les VOI.

2.2.3. Mirana : Un modèle à la fois compliqué et complexe

D'une part, Mirana est un modèle compliqué car, pour fonctionner, il considère de nombreux éléments du SES comme les institutions avec les différentes régulations, les inventaires des besoins et des ressources des foyers, les densités des ressources forestières, etc. D'autre part, il est aussi considéré comme étant un modèle complexe avec les interactions non linéaires existantes entre les nombreux éléments du modèle (interactions entre les régulations institutionnelles, interactions entre les ressources, etc.) et donc, le comportement global du modèle est non réductible à la composition des comportements individuels des éléments du modèle.

2.2.4. Construction et exploration du modèle Mirana

[3] http://mimosa.sourceforge.net/

Le modèle Mirana est construit sur la base d'un processus de modélisation formalisé par Farolfi [Farolfi et al. 2010]. Le processus de son développement est divisé en quatre étapes de spécification, à savoir :

- Le modèle conceptuel ;
- La dynamique des éléments du modèle ;
- L'initialisation du modèle ;
- L'observation du modèle.

2.2.4.1 Spécification du modèle conceptuel

Le modèle conceptuel décrit la structure du modèle de simulation avec la spécification de la terminologie utilisée dans le domaine de discours ainsi que les relations entre les termes utilisés. Dans MIMOSA [Müller 2004], le modèle conceptuel est basé sur les ontologies informatiques, plus particulièrement, la logique descriptive [Müller 2007]. La terminologie du modèle Mirana est basée sur six concepts clefs, à savoir : *les institutions, les territoires, les activités, les ressources, les populations et les organisations*. La Figure 1 représente la structure conceptuelle du modèle Mirana avec les relations entre les six concepts clés, sous forme de diagramme de classes UML (Unified Modeling Language).

Afin de montrer l'effort pour initialiser et observer un modèle de SES, nous allons décrire un à un les concepts clés du modèle Mirana avec les éléments nécessaires à son exploration.

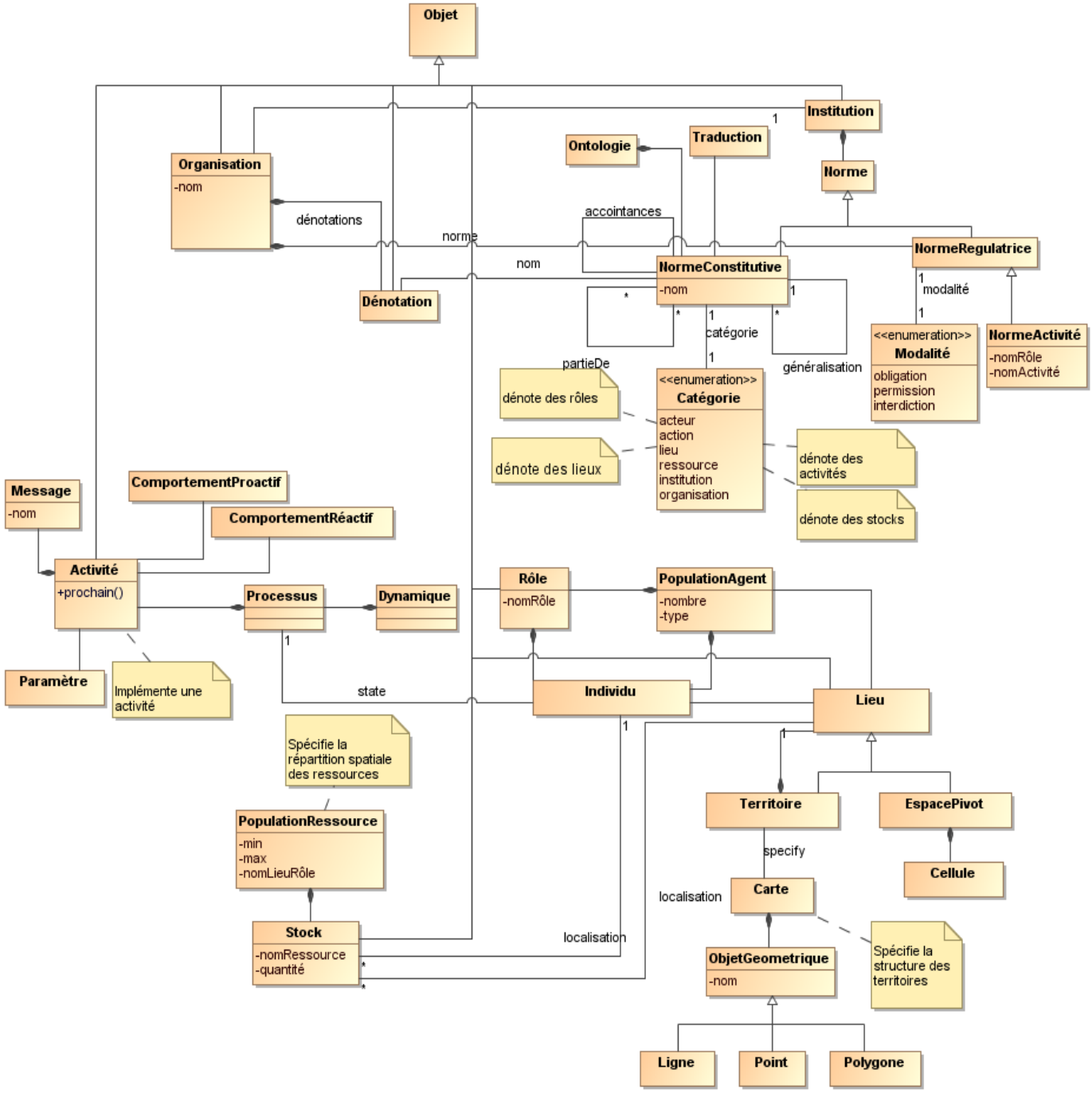


Figure 1. Modèle conceptuel du modèle Mirana

a. Les institutions

Les institutions (Figure 2) sont définies par un ensemble de normes, constitutives pour le vocabulaire et régulatrices pour les lois et les règles.

- Les normes constitutives définissent la terminologie sous la forme d'un ensemble de noms structurés par des relations de généralisation/spécialisation et des relations tout/partie. Ces

noms peuvent dénoter soit des objets (les identifiants), soit des ensembles d'objets (les concepts ou catégories), parmi lesquels on a les rôles des acteurs, les liens qui existent entre les acteurs, les lieux, les différentes ressources, etc.

- Les normes régulatrices énumèrent les permissions, obligations et interdictions. Ces normes portent sur la possibilité conditionnelle des activités des individus de se faire et de porter sur des objets, en des lieux et à des périodes. Elles sont formulées sur la base de la terminologie introduite par les normes constitutives.

Par ailleurs, d'autres éléments sont aussi associés aux institutions, à savoir :

- Des liens d'accointance qui lient éventuellement les rôles entre eux ;
- Des dénotations qui définissent quel(s) objet(s) désignent certains noms ;
- Des traductions entre noms de la même institution (synonymes) ou d'institutions différentes.

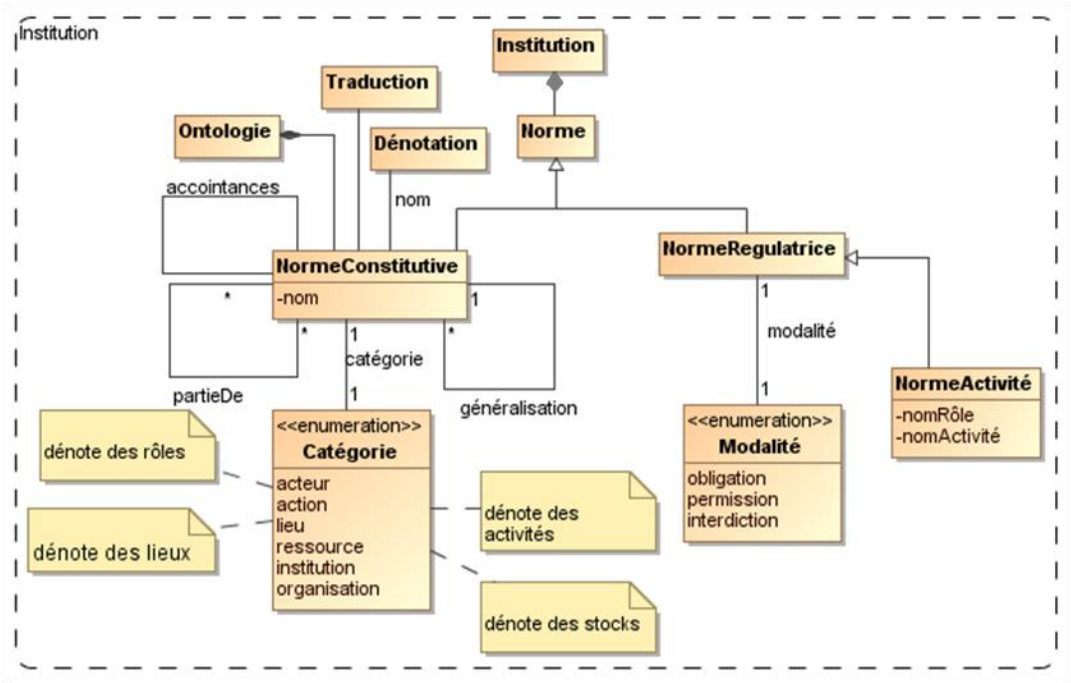


Figure 2. Structure conceptuelle d'une institution

Ainsi, pour décrire le modèle, il va falloir énumérer les institutions existantes comme par exemple, l'administration forestière, la commune, le foyer, la communauté locale de base, le marché légal et illégal, etc. puis les différentes normes constitutives qui leur sont associées. Pour cela, il est nécessaire de décrire et/ou de paramétrer, en plus de la liste des institutions connues :

- ***Les noms et les types d'objets qu'ils peuvent désigner***

  On distingue ainsi les rôles qui désignent des individus vus comme parties d'organisation,

les ressources qui désignent des stocks, les lieux qui désignent des territoires ou parties de territoire, les actions qui désignent des activités.

- ***Les hiérarchies entre les concepts***

  Un concept peut être un cas particulier d'un autre concept. Par exemple, dans une institution « Ecologie », un « Babakoto » est un cas particulier de « Animal » ;

- ***Les relations tout-parties entre les concepts***

  Un concept peut être une partie d'un autre concept. Par exemple, une « Zone Protégée » est une partie d'un « Zonage » ;

- ***Les relations que les acteurs entretiennent entre eux quand ils sont dans ces rôles respectifs***

  Dans le cas particulier, où les concepts désignent des acteurs, une relation peut être un concept d'un autre. Par exemple, la relation « Client » est un concept « Acheteur » du concept « Vendeur »;

- ***La traduction d'un concept d'une institution à une autre***

  Un concept défini pour une institution peut avoir un autre sens pour une autre institution. Par exemple, un « Gibier » pour un « Foyer » est un « Babakoto[4] » pour l'« Ecologie » ;

- ***L'implémentation des activités***

  Dans le modèle, les activités sont implémentées par des classes Java(au sens de la programmation orientée objet). Par exemple, pour que l'activité « Cultiver » des « Foyers » soit opérationnelle dans le modèle, il est nécessaire de préciser comme paramètre qu'elle soit implémentée par la classe Java « mirana.activities.Cultiver ».

Ensuite, il va falloir aussi décrire les normes régulatrices (i.e., les contraintes sur les éventuels processus). Dans le modèle, ces normes peuvent se présenter sous la forme de permissions ou d'obligations, telles que:

- ***La régulation de l'extension spatiale des activités des acteurs***

  Dans une institution « Institution », il est permis, si on a le rôle « Rôle », de faire l'activité « Activité » sur tel lieu « Lieu » d'un territoire « Territoire »;

[4] Une espèce de primate lémuriforme appartenant à la famille des Indriidés. C'est le plus gros des lémuriens, que l'on rencontre à Madagascar dans la forêt pluviale de la côte Est.

- ***La régulation des activités possibles pour produire des ressources***

  Dans une institution « Institution », il est permis, si on a le rôle « Rôle », de produire de la ressource « Ressource », de préférence « Préférence » par l'activité « Activité »;

- ***Les obligations de production***

  Dans une institution « Institution », il est obligatoire, si on a le rôle « Rôle », de produire de préférence « Préférence » telle quantité « Quantité », exprimée en unité « Unité », de ressource « Ressource » ;

- ***Les activités obligatoires***

  Dans une institution « Institution », il est obligatoire, si on a le rôle « Rôle », de faire de préférence « Préférence », l'activité « Activité ».

Les éléments mis entre guillemets représentent les informations nécessaires pour paramétrer le modèle, éventuellement, issues des données d'enquêtes décrivant les institutions en présence.

Ces normes permettent de définir les rôles respectifs des sujets de droit et des objets de droit, sachant que les sujets de droit sont des personnes physiques ou morales titulaires de droits et d'obligations et les objets de droit sont des objets sur lesquels portent des droits.

b. Les territoires

Les territoires jouent un grand rôle dans la définition du modèle à travers la nomenclature des lieux. Afin de faire correspondre cette nomenclature avec une géolocalisation précise, il faut définir l'ensemble des cartes auxquelles correspond cette nomenclature. Trois types de cartes sont disponibles dans la plateforme de modélisation, les cartes vectorielles, les cartes raster et les cartes pivots par combinaison des précédentes. Ces cartes sont vues comme un tout dont les objets géométriques de même nom sont les parties. Donc, un lieu pour une institution peut dénoter une carte auquel cas il s'appelle un territoire dont les parties sont elles-mêmes des lieux. Les lieux sont donc des noms pour des cartes ou des sous-ensembles des objets géométriques qui les constituent. Aussi, les cartes sont constituées [Aubert et Müller 2013] :

- d'un ensemble d'objets géométriques définis avec leur projection spatiale,
- de noms, pas forcément uniques, pour chacun de ces objets géométriques (on se donne ainsi le moyen de nommer un ensemble non vide d'objets géométriques).

L'union des objets géométriques que la carte contient est considérée comme un objet géométrique. La structure conceptuelle d'une carte est présentée sur la Figure 3.

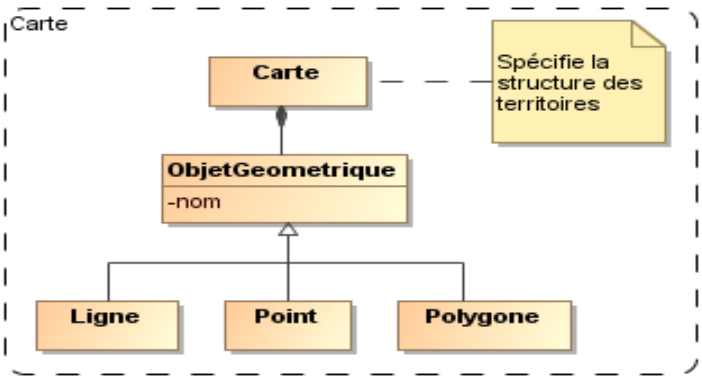


Figure 3. Structure conceptuelle d'une carte

c. Les activités

Les activités décrivent les différents comportements, de gestion (ex. gestion de contrats, gestion de permis, gestion de taxes, gestion du VOI, etc.) ou d'exploitation (ex. planification, plantation, production, vente de main d'œuvre, etc.) réalisés par les sujets de droit selon le rôle qu'ils jouent. Dans le modèle Mirana, les activités (Figure 4) sont associées à des classes Java qui implémentent les processus effectifs. Mais celles-ci sont éventuellement paramétrées par des tables qui spécifient les comportements en fonction du contexte organisationnel ou institutionnel dans lequel elles sont mises en œuvre. Les classes Java permettent d'implémenter notamment : un comportement réactif qui s'exécute lors de la survenue des événements isolément ou simultanément, un comportement proactif qui s'exécute spontanément au bout d'une durée déterminée, ainsi que le calcul de durée jusqu'au prochain comportement proactif. La structure conceptuelle des activités est représentée sur la Figure 4.

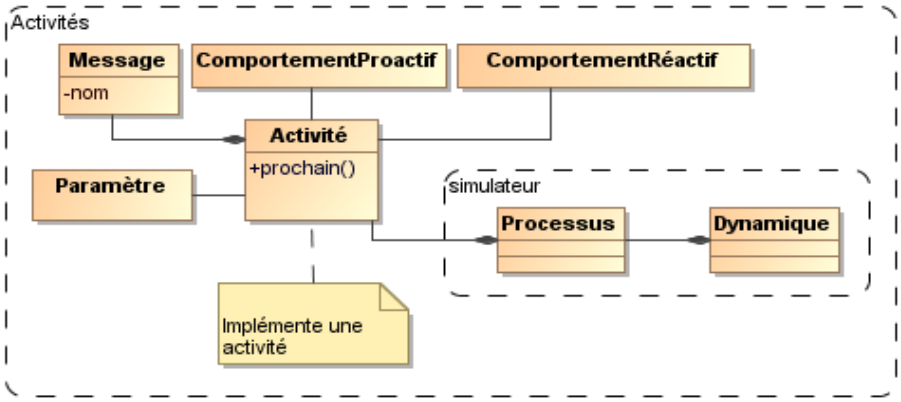


Figure 4. Structure conceptuelle des activités

Par conséquent, il va falloir énumérer les activités avec les évènements auxquels le comportement réagit et qu'il engendre, afin de paramétrer chacune d'elles.

Considérons quelques exemples d'activités :

- ***La croissance des ressources***

  Les ressources sont soumises à différentes dynamiques intrinsèques comme la croissance, la mobilité. Dans le modèle, ces dynamiques ne sont pas spontanées, mais mises en œuvre selon des points de vue spécifiques. L'écologue, représenté par un agent, aura en charge les dynamiques des ressources écologiques, le démographe aura en charge les dynamiques démographiques. Ainsi les points de vue disciplinaires sont intégrés dans le modèle et participent de sa dynamique. Et afin de paramétrer la croissance des ressources, il faut définir l'institution dans laquelle elle prend sens, le territoire sur lequel agit cette activité de croissance, la ressource sur laquelle elle porte et enfin les taux de croissance eux-mêmes.

- ***Les déplacements***

  La pénétrabilité du territoire varie en fonction des infrastructures et du mode d'occupation du sol. Dans le modèle, les distances sont exprimées en durée de parcours afin d'apprécier les coûts de transaction pour accéder aux ressources. Pour définir les distances, il est nécessaire de fournir le nom de la carte, le nom du lieu et la durée de parcours selon les lieux (forêt, piste, etc.).

- ***Les permis***

  Dans le modèle Mirana, les membres du VOI demandent systématiquement un permis d'usage pour satisfaire leurs besoins autres que financiers et permis d'exploitation pour leurs besoins financiers. Les permis sont octroyés en fonction des quotas, naturellement associés à l'organisation correspondante (et pas à l'institution). Aussi, pour paramétrer les permis, nous avons besoin de définir l'organisation pour laquelle le quota est défini, le territoire et le lieu du territoire sur lequel le quota s'applique, la ressource concernée, la quantité maximum utilisable ou exploitable et l'unité correspondante.

- ***Les transports de ressources***

  Il s'agit de définir la quantité de ressources qu'un agent peut transporter par voyage en fonction du moyen de transport utilisé. Pour cela, il est nécessaire de spécifier, le nom de l'‘institution dans laquelle la ressource et le moyen sont définis, le nom de la ressource à transporter, le moyen de locomotion, la quantité par voyage et l'unité correspondante.

- ***La culture***

  La culture en tant qu'exploitation des ressources utilise surtout de la surface, de la fertilité et de l'eau. Mais dans le modèle, seul l'itinéraire technique, à savoir le rendement attendu

et le temps de travail (en hommes/jour) pour les différentes phases de la culture, et les informations guidant la prise de décision dans une version très simplifiée, sont considérés. Pour le paramétrage, il faut spécifier l'institution dans laquelle la ressource est définie, la dénomination de la ressource qui correspond à une variété, le rendement attendu en kilogramme par hectare ($kg.ha^{-1}$), le lieu où la variété est cultivable, le temps de préparation du terrain en hommes par jour par hectare préparé, le temps de semis en hommes par jour par hectare semé et le temps de récolte en hommes par jour par hectare récolté.

- ***Le versement de taxes***

  La vente de certains produits sur le marché par un membre de VOI implique le versement de taxes concrétisé par le transfert d'une somme d'argent du patrimoine d'un sujet de droit (foyer par exemple) à un autre (gestionnaire par exemple).Les taxes sont définies par institution. Aussi, pour paramétrer le versement de taxes, il est nécessaire de spécifier l'institution qui applique la taxation, l'institution taxée, le rôle qui autorise à percevoir les taxes, le produit à taxer, la quantité taxée sous la forme de son unité, le montant de la taxe par unité de la ressource et l'unité du prix.

- ***Le contrôle***

  Les différentes organisations définissent des contraintes très diverses sur les activités. Il se pose alors le problème du contrôle réalisé pour une activité ou un ensemble d'activités spécifiques. En cas de constat de violation, il y a lieu de définir les amendes à appliquer. A cet effet, il est nécessaire de paramétrer les amendes applicables selon la ressource sur laquelle a été constatée la violation. Pour ce faire, il faut préciser l'institution en charge de la collecte des amendes, le rôle qui autorise la collecte des amendes, la ressource concernée, le montant de l'amende et l'unité correspondante.

d. Les ressources

Les ressources dont la structure est présentée sur la Figure 5 sont des objets de droits appropriables par des sujets de droits. Elles sont appréhendées par la notion de stocks qui sont représentés par des quantités. Les stocks sont répartis dans des lieux ou chez les individus. Les lieux permettent de localiser et de différencier les types de stocks de ressources. Les ressources sont réparties sur le fond de terre selon une densité à l'hectare et les actions spatialisées sont normalisées à l'hectare. Un fond de terre est la surface élémentaire faisant sens pour les actions spatialisées des individus. Il peut varier d'une simulation à l'autre.

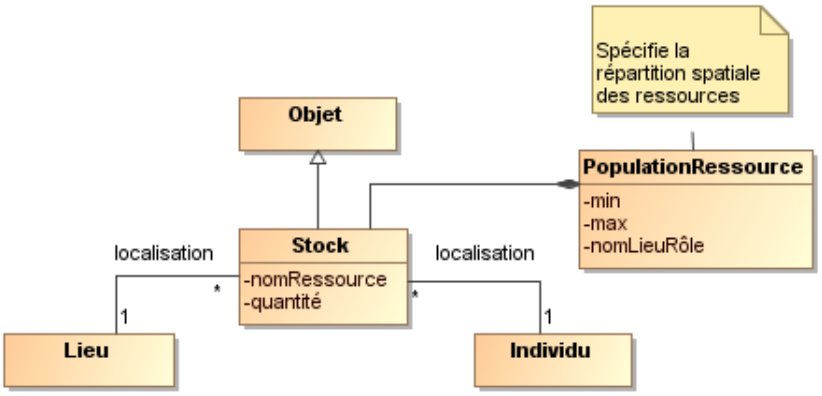


Figure 5. Structure conceptuelle de la ressource

Ainsi, pour décrire le modèle, il faut connaître ses densités. Pour ce faire, il faut fournir, d'une part, les informations sur les ressources, i.e., le nom de l'institution dans laquelle les noms de la ressource, du territoire et du lieu sont définis, le nom de la ressource, le nom du territoire ou de l'organisation, le lieu du territoire ou le rôle de l'organisation, la quantité minimum et la quantité maximum, l'unité des quantités qui précèdent, et la densité. D'autre part, il faut aussi spécifier les contraintes supplémentaires de la localisation des ressources sur les territoires : par exemple, la quantité de poissons dépend de la présence d'une rivière ou d'un plan d'eau, mais aussi de la végétation environnante.

e. Les organisations

Une organisation dont la structure est représentée par la Figure 6 est considérée comme étant un groupe d'individus qui met en œuvre une institution en se partageant les rôles au sein de cette institution, impliquant une compréhension commune de sa terminologie [Ostrom 2009]. Les organisations entretiennent des liens entre elles à travers les rôles de leurs membres respectifs. Un individu peut appartenir à différentes organisations. Ainsi, les organisations se superposent avec l'ensemble des individus.

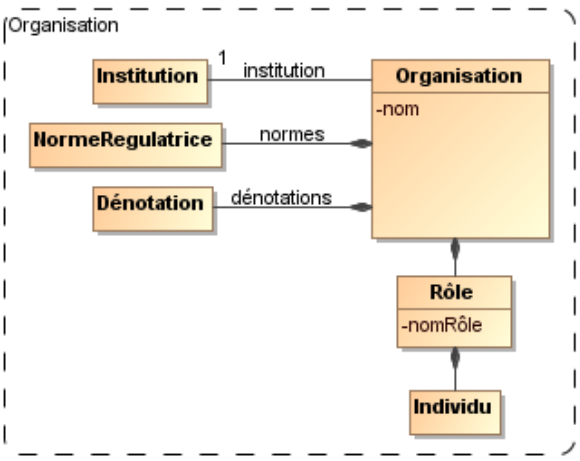


Figure 6. Structure d'une Organisation

Par conséquent, pour décrire le modèle, il va falloir énumérer les organisations en précisant:

- L'institution dont elles constituent la mise en œuvre ;
- Les rôles que les individus jouent dans l'organisation, tels qu'ils sont définis par l'institution et dénotant l'ensemble des individus jouant le rôle dans cette organisation ;
- Des dénotations qui définissent quel(s) objet(s) désignent certains noms de l'institution pour cette organisation ;
- Des normes régulatrices qui énumèrent les permissions, obligations et interdictions spécifiques à cette organisation, en plus de celles de l'institution mises en œuvre par l'organisation.
- Les accointances entre rôles de la même organisation (intra-organisationnelle) ou d'organisations différentes (inter-organisationnelles). Elles s'ajoutent aux accointances intra-organisationnelles spécifiées par l'institution mise en œuvre.

f. Les populations

Une population dont la structure est représentée sur la Figure 7, est un groupe d'individus qui partagent la même localisation spatiale (éventuellement aucune) et les mêmes appartenances à des organisations. L'ensemble des populations constitue l'ensemble des individus du système, tel que chaque individu met en œuvre les activités qui lui échoient dans le cadre de ses multiples appartenances sociales.

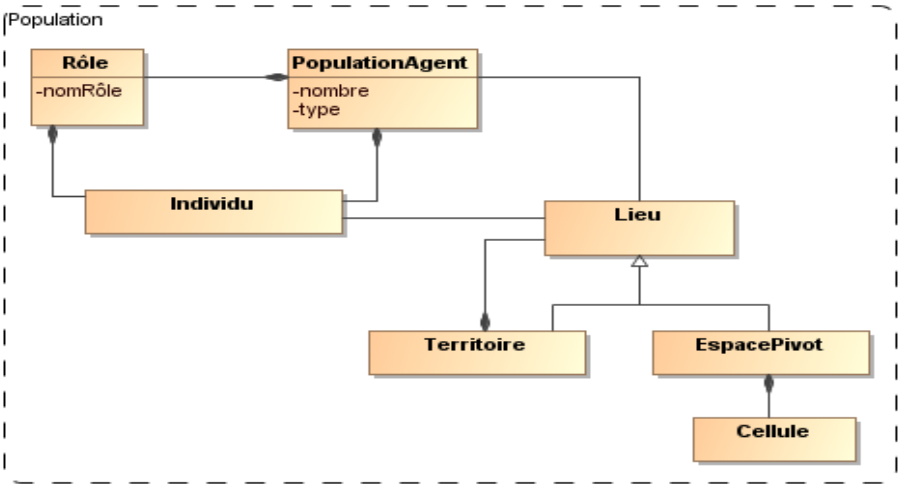


Figure 7. Structure conceptuelle d'une population

Ainsi, pour paramétrer le modèle, il va falloir faire des recensements de la population avec les appartenances sociales de cette dernière afin d'obtenir le nombre d'individus, le lieu sur lequel ces derniers se trouvent éventuellement (tels que des individus peuvent ne pas avoir de localisation géographique) et les rôles qu'ils jouent dans telles ou telles organisations.

2.2.4.2 Spécification du comportement des éléments du modèle

Après la spécification du modèle conceptuel et le paramétrage des différents éléments du modèle, les dynamiques permettant de faire évoluer le modèle sont mises en œuvre et doivent aussi être paramétrées. Ces dynamiques, ou processus d'évolution, sont basées sur le formalisme DEVS (pour Discrete Event Simulation ou Simulation à Evènement Discret) [Zeigler et al. 2000] permettant la modélisation, la simulation et l'analyse des systèmes complexes développés.

Pendant la simulation, ces dynamiques produisent un changement d'état, spontané et réactif des éléments du modèle (sujets de droit ou objets de droit), lors des interactions entre les dynamiques sociales et biophysiques. Les interactions issues de ces dynamiques engendrent, en général, des trajectoires imprévisibles du modèle, et déclenchent, de la part des thématiciens, des débats et des discussions sur le fonctionnement du modèle.

2.2.4.3 Initialisation du modèle Mirana

L'initialisation du modèle consiste à définir un scénario, c'est-à-dire, à construire l'état initial du modèle, à paramétrer les processus (ou les comportements) et à spécifier éventuellement des séries temporelles (par exemple, les séries climatiques).

Nous avons montré dans ce qui précède que nous avons besoin d'une grande quantité de données et d'informations sur le SES en question. Ces informations dont on a besoin peuvent provenir, de diverses sources issues de projets traitant des thématiques différentes, par exemple :

- Le logiciel « Olympe » qui renferme un ensemble de données sur les exploitations agricoles. Il a été développé par l'INRA/ESR en collaboration avec l'IAMM et le CIRAD [Penot 2012] et permet de faire de la modélisation et de la simulation du fonctionnement de l'exploitation agricole. Ce logiciel peut, par exemple, fournir les données nécessaires pour le paramétrage des itinéraires techniques du modèle Mirana, i.e. le rendement attendu d'une culture ainsi que le temps de travail pour les différentes phases de culture et éventuellement, la surface et la fertilité du sol.
- Le projet « Patrimoine Numérique » qui est en cours de réflexion au niveau du CIRAD[5]et qui a pour objectif de référencier toutes les données scientifiques produites au sein du CIRAD, y compris les données secondaires issues de la littérature. Les données référencées dans cette plateforme peuvent être utilisées pour réaliser les inventaires sur les besoins et les ressources des foyers et/ou pour initialiser les densités des ressources forestières du modèle Mirana par exemple.
- Le réseau GBIF[6] (pour Global Biodiverity Information Facility ou en français, Système Mondial d'Information sur la Biodiversité) [Yesson et al. 2007] suscité de type « méga-science », est une banque de données sous tutelle du gouvernement français, en partenariat avec plusieurs organismes de recherche, permet de publier des données sur la biodiversité dans le monde entier. Et il permet surtout de les exploiter gratuitement dans le but de mettre à la disposition des scientifiques et du grand public les informations sur la biodiversité mondiale afin de pouvoir assurer un avenir scientifique, écologique, social cohérent et durable. Cette banque de données permet par exemple de fournir les cartes thématiques dont on a besoin avec une géolocalisation précise des espèces et des ressources prises en compte dans le modèle Mirana.
- Les démarches de recherches menées par les thématiciens comme les enquêtes sociales ou les inventaires ainsi que les données secondaires issues de la littérature. Ces données et ces informations peuvent être utilisées pour décrire et paramétrer, par exemple, les activités des foyers ou la croissance de la population du modèle.

[5]http://ur-agirs.cirad.fr/agenda-evenements/2014/quel-projet-pour-le-patrimoine-numerique-scientifique-du-cirad
[6]http://www.gbif.fr/

Ces données sont, en général, sauvegardées sur des supports de données et présentées sous différents formats comme des fichiers Excel, des fichiers CSV, des fichiers shapefiles avec des tables attributaires pour les cartes, des bases de données relationnelles, du OWL (Web Ontology Language), des composants d'une interface graphique d'un logiciel, etc.

En plus de la quantité de données et la diversité des sources, les structures de données recueillies sont hétérogènes et nécessitent généralement des transformations ou des compositions plus ou moins compliquées pour les adapter aux structures de données informatiques utilisées dans le modèle. A titre d'exemple, nous avons étudié la façon dont le modèle IMAS [Belem et al. 2011a] exploite les données des thématiciens pour fonctionner. En effet, IMAS est un modèle de simulation multi-agents développé au sein de notre équipe sur la plateforme MIMOSA [Muller 2004], pour tester des scénarios d'évolution de la diversité variétale au Chili et au Mali. Son utilisation pose un problème pour les thématiciens car les passages des données représentées sur la Figure 8, dont ils disposent vers la base de données de simulation représentée sur la Figure 9 conçue par l'informaticien/modélisateur, ainsi que de la base de données de simulation (Figure 9) vers le modèle conceptuel représenté sur la Figure 10, sont considérés comme des « boîtes noires » pour eux. Nous avons constaté différents types de transformation pour passer des données des thématiciens vers le modèle conceptuel afin d'initialiser et paramétrer le modèle, à savoir : des fusions, des créations, des modifications, des suppressions et même des changements de types d'entité ou d'attributs d'entité. Et plus encore, tous ces changements ne sont pas accessibles aux utilisateurs du modèle de simulation, ce qui ne permet pas de les valider.

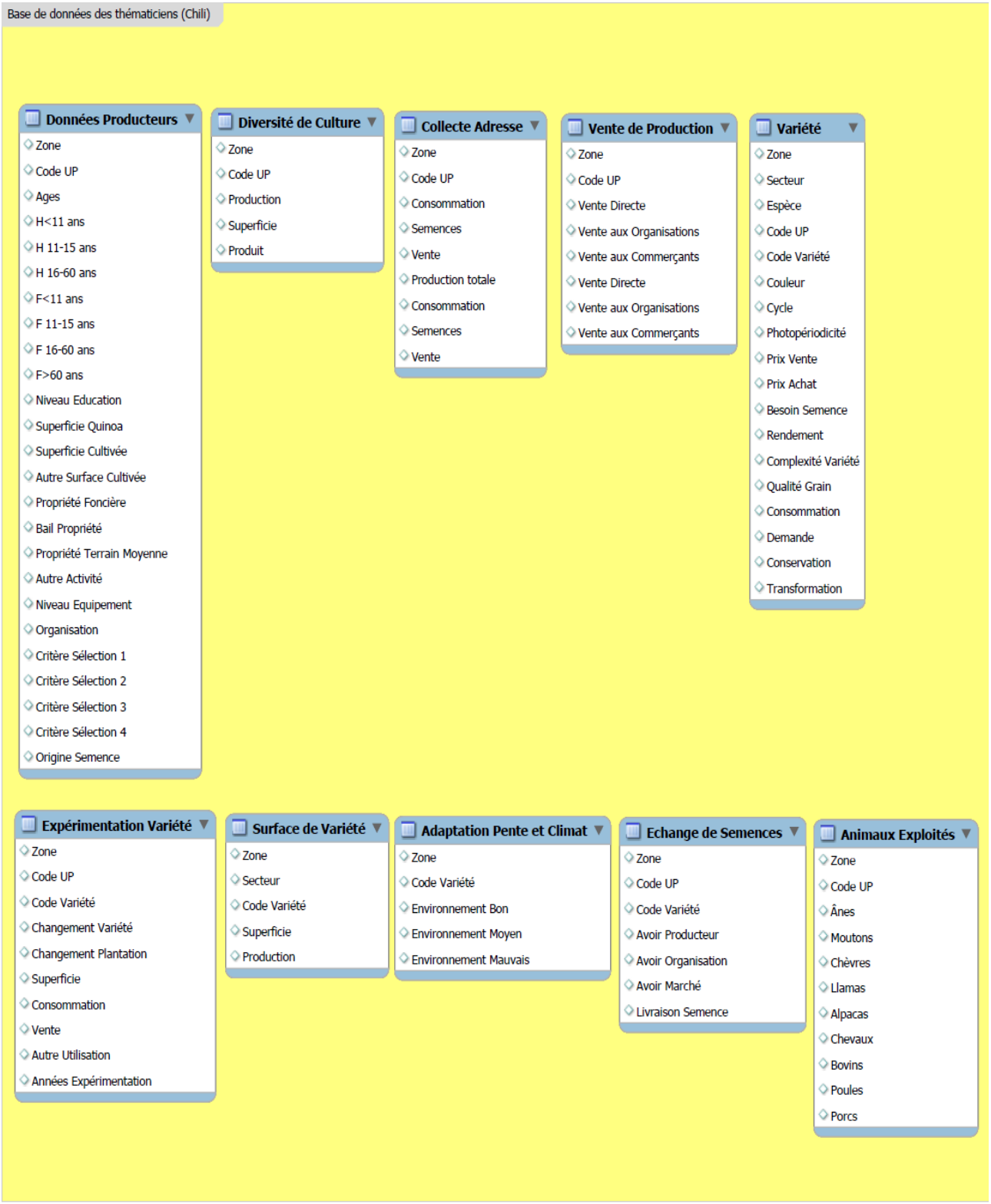


Figure 8. Base de données des thématiciens du projet IMAS [Belem et al. 2011a] Cette base de données sur le système semencier est élaborée à partir de différents sondages et enquêtes au Chili. Cependant, sa structure n'est pas celle qui est utilisée par le modélisateur dans l'implémentation informatique.

Figure 9. Base de données de simulation de IMAS - version Octobre 2011[Belem et al. 2011a]

Cette base de données est construite par le modélisateur. Elle évolue souvent et dépend des contraintes techniques rencontrées lors de l'implémentation du modèle. Par conséquent, elle ne correspond pas souvent à la base de données des thématiciens (Figure 8). A titre d'exemple, pour le projet IMAS, nous avons trouvé trois différentes versions de base de données de simulation. La première version date de Mars 2011, la deuxième de Septembre 2011 et la dernière date d'Octobre 2011.

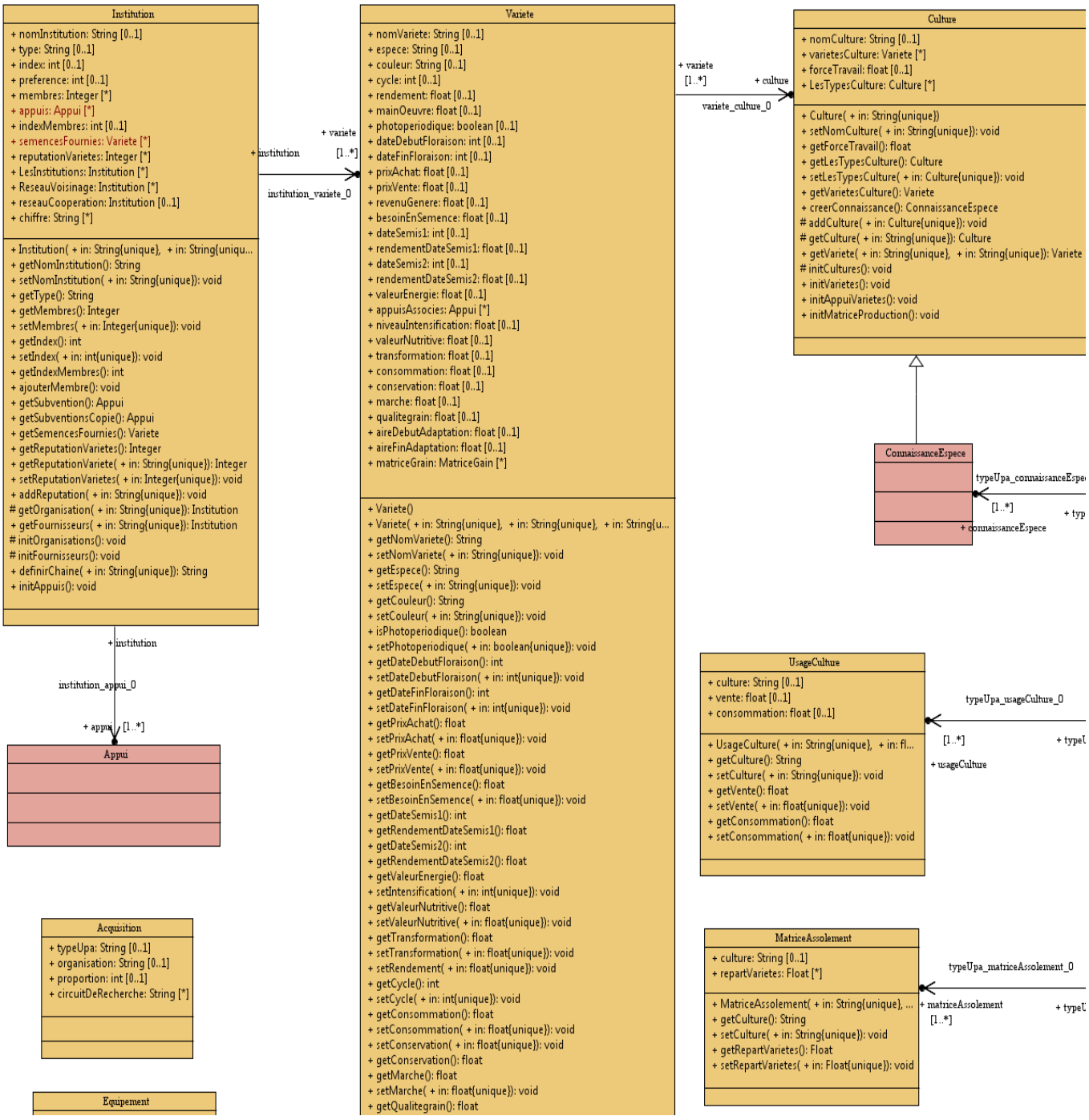


Figure 10. Extrait du modèle conceptuel du modèle IMAS[7]

L'initialisation du modèle de simulation consiste à instancier ce modèle conceptuel avec les données issues de la base de données "transformée" des thématiciens (i.e. Figure 8 vers Figure 9).

[7] Reverse engineering réalisé par Diallo Abdoulaye, un doctorant au sein de l'équipe GREEN

En résumé, à part les transformations nécessaires à l'initialisation de modèle, Mirana comme IMAS nécessite différents types de données et d'informations qui peuvent provenir de différentes sources hétérogènes, telles que :

- Les données sur les institutions, les organisations ainsi que les activités, essentiellement issues des données d'enquêtes réalisées auprès des communes, des régions ou de la zone d'études ;
- Les données sur les territoires obtenues à partir des cartes thématiques (vecteur ou raster) fournies par les géographes ;
- Les données sur les ressources et les populations extraites à partir d'un ensemble de données issues de différentes sources et de différents projets (par exemple, le logiciel Olympe [Penot 2012], le patrimoine numérique du CIRAD, le GBIF [Yesson et al. 2007], d'une interface graphique quelconque d'un logiciel, ou autres).

2.2.4.4 Observation du modèle Mirana

Cette phase consiste à spécifier ce qu'on voudrait observer dans le modèle pendant la simulation. L'observation du modèle consiste à obtenir et à présenter l'état et l'évolution du modèle pendant la simulation et en élaborer des indicateurs[8] synthétiques sur des données identifiables (revenus, diversité floristique et faunistique, coût du contrôle, répartition spatiale des ressources, etc.) de façon proche des besoins des thématiciens et des acteurs locaux. Ils sont relatifs à des questions de compréhension du modèle, de recherche ou de développement. Sa spécification permet à la simulation, de répondre aux questions posées par les thématiciens sur le SES étudié.

Pour comprendre les besoins en observation, supposons que nous voulons avoir des réponses issues de la simulation du modèle sur la question suivante, posée par les thématiciens : « Quel peut-être l'impact de la mise en place des systèmes de régulation sur la durabilité écologique, économique et sociale d'un terroir villageois à Madagascar ? »

Nous avons constaté, en discutant avec les thématiciens, que les besoins sont nombreux et peuvent varier d'une simulation à l'autre, à savoir :d'une part, les besoins d'avoir un moyen permettant d'identifier, de suivre et de manipuler certains éléments observables, tels que par exemple:

- ***Pour comprendre la résilience des espèces***, il s'agit de suivre tout au long de la simulation, la quantité totale de ressources prélevées (sous-partie du modèle) puis de la

[8]Les indicateurs sont des indices ou un résumé d'informations définis et très utilisés par les thématiciens pour pouvoir mesurer l'évolution de différents éléments qui les intéressent dans un SES à des fins d'évaluation et/ou d'aide à la prise de décision.

comparer avec le maximum utilisable de chaque ressource afin d'évaluer son renouvellement (sous-partie agrégée du modèle);

- ***Pour comprendre le processus de conservation de la biodiversité*** qui contribue à la durabilité écologique, il s'agit de suivre l'évolution de la surface de chaque type d'habitat (sous-partie du modèle) ou l'évolution de la population floristique et faunistique totale (sous-partie agrégée du modèle) ;
- ***Pour comprendre l'enrichissement ou l'appauvrissement des foyers*** qui contribue à la durabilité économique et sociale, il s'agit de suivre l'évolution de la satisfaction des besoins de chaque foyer (sous-partie du modèle) ou l'évolution moyenne de la satisfaction des besoins des foyers dans un village (sous-partie agrégée du modèle) ;
- ***Pour comprendre les interactions dans le modèle***, il s'agit par exemple de suivre le nombre de permis demandés et acceptés auprès des gestionnaires (sous-partie du modèle), le pourcentage du nombre de permis demandés et acceptés (sous-partie agrégée du modèle) ou l'évolution du nombre d'activités illégales sur un territoire ;
- ***Pour comprendre le coût de conservation***, il s'agit de calculer le coût (sous-partie agrégée du modèle) à partir du nombre de contrats de surveillance délivrés par l'administration forestière (sous-partie du modèle) ;
- ***Pour comprendre la répartition des ressources***, il s'agit par exemple d'extraire la quantité (sous-partie du modèle) ou la moyenne théorique cumulée (sous-partie agrégée du modèle) de chaque type de ressource sur le territoire du VOI, la quantité ou la moyenne totale effectivement cumulée sur le territoire du VOI, ou encore la répartition de cette quantité ou moyenne cumulée par couvert végétal ;
- ***Pour l'analyse individuelle***, il s'agit par exemple de suivre ce que fait chaque acteur (sous-partie du modèle) du système dans le cadre d'une activité spécifique.

Et d'autre part, les besoins de spécifier une stratégie d'observation afin de pouvoir constater ce qui se passe dans le modèle de simulation à une date précise ou selon un pas de temps constant (tous les mois ou toutes les années par exemple) ou seulement au cas où il y a changement d'état des observables du modèle.

En effet, ces besoins se résument par la possibilité de spécifier:

- L'observation d'une sous-partie du modèle, pour pouvoir suivre un certain nombre d'éléments observables du modèle pendant la simulation;

- L’observation agrégée sur une sous-partie du modèle, pour des analyses plus approfondies, par exemple, le nombre moyen, le nombre total, le quantile, la variance ou l’écart type d’un élément observable du modèle afin d’obtenir un autre type d’indicateur plus agrégé par rapport aux éléments observables du modèle;
- Des stratégies d’observation pour définir le moment ou la fréquence d’observation du modèle pendant la simulation.

A part l’identification et la manipulation des éléments observables du modèle pour produire les indicateurs souhaités, il faut aussi avoir un moyen de les visualiser à l’écran en se servant des différents composants visuels disponibles tels que les histogrammes, les courbes, les grilles, les zones de textes, les étiquettes, les cartes, les tableaux, etc.et/ou de les sauvegarder dans des puits de données à des fins d’éventuelles analyses ou réflexions. A titre d'exemple, la Figure 11 présente visuellement un résultat de simulation sur une espèce d’arbre. La liste comprend à la fois les ressources primaires (les espèces) et les produits secondaires (grumes, etc.). Plus particulièrement, nous pouvons constater qu’il y a entre 0 et 11 varongy (Ocatea tricophlebia) par parcelle (1/4 d’hectare dans ce cas) avec une forte déplétion autour des villages.

Par ailleurs, il est aussi possible que les thématiciens posent des questions plus détaillées chemin faisant. Par exemple : « Pourquoi un village donné n’aboutit pas aux mêmes résultats de simulation qu’un autre ? », ou « Pourquoi un foyer ne commet jamais d’actions illégales ? » ou « Pourquoi le *tavy* ne cause pas autant de déplétion à partir de la simulation? » etc. Il devient ainsi nécessaire d’avoir des moyens pour exploiter encore la simulation afin d’améliorer la compréhension des résultats d’une simulation et de répondre aux nouvelles questions qui émergent de celle-ci.

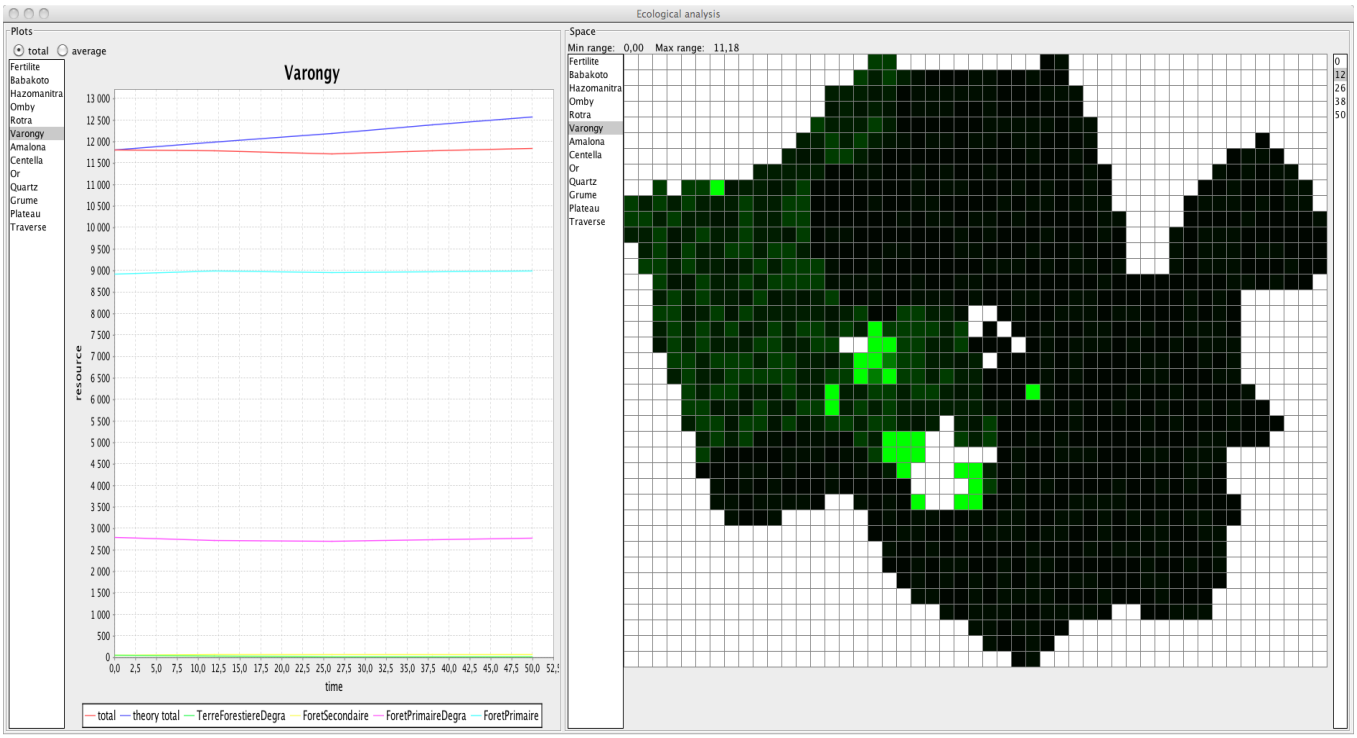


Figure 11. Les indicateurs écologiques (Evolution mensuelle du nombre d'individus)

2.3. Conclusion

Nous avons montré dans ce chapitre, par la présentation du modèle de simulation « Mirana », que les modèles de SES sont compliqués et complexes. Pour fonctionner, ils nécessitent d'une part, une multitude de données et d'informations issues de sources de données hétérogènes. Et d'autre part, les interactions entre les éléments du modèle pendant la simulation sont imprévisibles. Par conséquent, l'exploration des modèles de SES devient difficile et demande un effort considérable. Le processus d'initialisation nécessite beaucoup de codage de la part des informaticiens et des modélisateurs afin d'extraire les données recueillies et de les transformer en des structures de données adaptées aux modèles de simulation. Il est aussi difficile de satisfaire les besoins des thématiciens en termes d'observation de modèle de SES ce qui nécessite aussi beaucoup de codage. Ces besoins sont liés à leurs intérêts qui sont relatifs à des questions de compréhension du modèle, de recherche ou de développement. De ce qui précède, il devient un besoin important de pouvoir résoudre le problème de l'initialisation et de l'observation des modèles de SES afin de faciliter son exploration.

Aussi, deux questions se posent principalement, à savoir la manière d'exploiter efficacement les nombreuses données hétérogènes en possession des thématiciens pour faciliter l'initialisation des

modèles de SES et celle de construire à partir du modèle de simulation, des indicateurs relatifs à la fois, aux questions des thématiciens et à la compréhension de ces modèles.

Le chapitre suivant traitera par conséquent, de l'état de l'art sur ce qui a été fait pour traiter les problèmes d'initialisation et d'observation des modèles. Rappelons que l'objectif de la présente étude consiste à élaborer une méthodologie générique du processus d'initialisation et d'observation de modèles de SES, adaptée aussi bien aux thématiciens qu'aux modélisateurs.

Chapitre III. Etat de l'art

3.1 Introduction

Nous avons montré que le processus d'initialisation et d'observation de modèles est nécessaire et déterminant dans la démarche de modélisation et de simulation (M&S). D'une part, pour qu'un modèle de simulation puisse fonctionner, son initialisation est incontournable. Initialiser un modèle revient, en général, à créer des objets (au sens de la programmation orientée objet) ou des structures de données conformes au modèle, avec leurs éventuelles relations, puis à définir les valeurs des attributs, ou des propriétés, respectifs pour construire son état initial. D'autre part, une fois le modèle initialisé, afin de pouvoir suivre et faire des analyses prospectives sur ce qui se passe pendant la simulation, avoir un moyen permettant d'observer ce qui se passe dans le modèle devient aussi nécessaire. Observer un modèle revient à pouvoir extraire et présenter des informations, des traces de simulation ou des indicateurs spécifiques définis par les thématiciens, à partir de la simulation.

Des recherches permettant d'explorer des modèles de simulation ont été effectuées et ont donné lieu à certaines réalisations informatiques comme le logiciel « BehaviourSpace » intégré dans la plateforme de M&S Netlogo [Tisue et Wilensky 2004], SimExplorer [Amblard et al. 2003] et OpenMOLE [Reuillon et al. 2013].

- **BehaviourSpace** [Graham 2009] : un logiciel intégré dans la plateforme NetLogo qui offre une interface graphique pour spécifier des expérimentations[9]. Cette interface permet de configurer une succession de simulations afin de construire dynamiquement des jeux de scénarii selon les besoins du modélisateur. Le logiciel offre les moyens pour spécifier des sous-ensembles de valeurs possibles de chaque paramètre utilisé par le modèle lors de la simulation.

  Par conséquent, l'utilisateur a la possibilité de :

  - Définir, avec des formules ou des équations, les données (ou indicateurs) que les thématiciens veulent récolter et suivre parmi les différents observables (variables et paramètres) du modèle ;
  - Répéter plusieurs fois une simulation avec les mêmes configurations en sachant que le comportement d'un modèle SMA peut toujours varier d'une simulation à l'autre ;

[9] Expérimentation : succession de simulations avec, en général, des paramètres initiaux différents pour chaque simulation [http://www.esmoutier.ch/docs_eleves/NetLogo_ESM/docs/behaviorspace.htm].

- Spécifier une stratégie d'observation en précisant à quel moment de la simulation l'outil va mesurer et présenter les données de sorties du modèle, i.e. est-ce qu'il va récupérer les observables, par exemple, à chaque pas (ou tick) de simulation, périodiquement (tous les 100 pas par exemple) ou seulement à chaque fin de simulation;

- Choisir le format dans lequel les données récupérées seront sauvegardées. Deux formats sont disponibles dans NetLogo : le format « table », où les données récupérées sont recueillies et écrites dans un fichier de sortie standard CSV (Comma Separated Values) à la fin de la simulation, et le format « spreadsheet » permettant de calculer les valeurs minimale, maximale, moyenne et finale de chaque paramètre mesuré et de les enregistrer en CSV seulement à la fin de l'expérimentation.

Bien que le logiciel *BehaviorSpace* englobe tout le processus de modélisation, en partant du paramétrage des expérimentations d'un modèle de simulation jusqu'à la génération de plusieurs sorties successives de simulations, il ne traite pas directement les problèmes d'initialisation et d'observation. Rappelons que l'initialisation du modèle d'une part consiste à décrire les données à la disposition des thématiciens, à spécifier les structures de données puis à les transformer jusqu'à ce qu'elles correspondent à celles du modèle de simulation. C'est seulement après qu'il est possible de générer l'état initial du modèle à partir des données externes. L'observation de modèle d'autre part consiste à pouvoir récupérer des sous-parties des observables du modèle selon une stratégie d'observation, puis à les manipuler afin d'obtenir et de présenter des indicateurs que les thématiciens souhaitent observer pendant la simulation.

En ce qui concerne l'initialisation de modèle, *BehaviorSpace* ne permet pas d'utiliser les données stockées sur des supports externes pour le paramétrage des simulations successives. Il génère un certain nombre de valeurs, selon une spécification faite par l'utilisateur sous forme d'intervalle de valeurs, mais ne se préoccupe pas de la manière dont ces valeurs seront exploitées pour pouvoir initialiser le modèle de simulation. Ceci doit être codé en dur dans la plateforme de M&S.

En ce qui concerne l'observation de modèles, la récolte des données pendant la simulation dépend d'un compteur à pas de temps constant appelé « ticks » intégré dans NetLogo, et le logiciel *BehaviorSpace* ne fournit pas le moyen pour spécifier d'autres types de stratégie

d'observation comme par exemple, la possibilité de récupérer l'état d'un élément du modèle en cas de son changement d'état.

L'utilisation de *BehaviorSpace* met ainsi en évidence la nécessité d'avoir un moyen permettant de décrire ou de spécifier:

- Non seulement des intervalles de valeurs pour paramétrer un modèle de simulation, mais aussi des données externes;
- Le passage des données vers le modèle de simulation;
- Des équations ou des formules statistiques manipulant les observables du modèle de simulation afin d'observer les indicateurs souhaités par les thématiciens;
- Des stratégies d'observation selon les besoins des thématiciens ;
- La présentation visuelle des sorties souhaitées de la simulation.

- **SimExplorer** [Amblard et al. 2003] : un logiciel utilisé pour explorer des systèmes multi-agents. Il permet de :
  - Générer et distribuer des expériences de simulation selon un ensemble de paramètres d'entrée spécifiés par l'utilisateur ;
  - Récupérer, traiter et présenter des résultats sous forme d'indicateurs ;
  - Exécuter des simulations sur une machine locale, sur un cluster ou sur une grille de calcul.

  En effet, après les différentes spécifications faites par l'utilisateur, l'outil se charge de générer automatiquement des expériences, en exécutant les simulations soit sur une machine soit sur une grappe de machine dans un réseau local, en distribuant les exécutions sous forme de modules (M1, M2, M3, M4 de la Figure 12). Les résultats des simulations peuvent être ensuite récupérés et traités (avec des calculs statistiques pour synthétiser les résultats par exemple) puis visualisés graphiquement à l'écran.

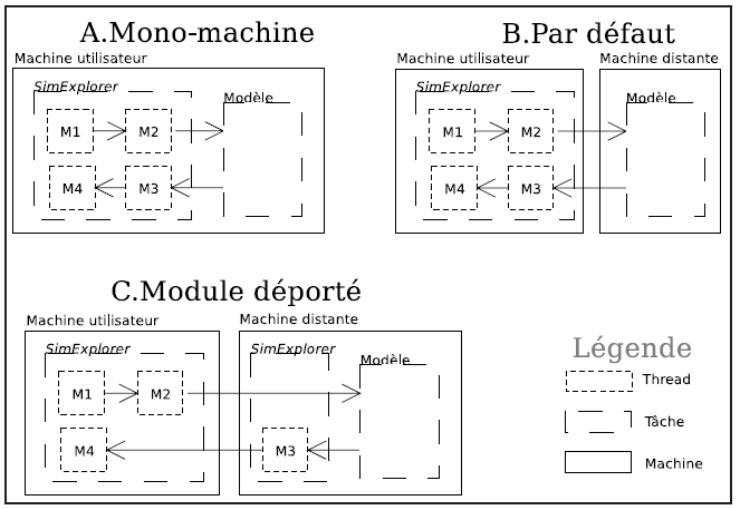


Figure 12. Trois façons différentes d'utiliser SimExplorer [Bernard Stéphan 2004]

Bien que *SimExplorer* soit un environnement complet pour explorer les modèles multi-agents, son utilisation reste compliquée et il est difficile à prendre en main par les thématiciens. Néanmoins, nous avons pu identifier aussi des éléments essentiels pour pouvoir initialiser et observer les modèles de simulation, à savoir :

- Pour l'initialisation de modèle, *SimExplorer* suscite la nécessité d'avoir un moyen de décrire les sources de données d'entrée du modèle en précisant le type du fichier et son chemin, puis de spécifier la façon dont les données seront exploitées pour générer les entrées du modèle ;
- Pour l'observation de modèle, comme les simulations peuvent éventuellement être distribuées sur plusieurs machines, *SimExplorer* n'offre pas spécialement de stratégie d'observation, mais sauvegarde toutes les données et toutes les traces de simulation sur un serveur. Ensuite, il met à disposition de l'utilisateur un moyen permettant d'exploiter les sorties des simulations pour calculer les indicateurs. Nous pouvons nous inspirer de son environnement de développement pour récupérer et traiter les sorties du modèle afin de construire et de présenter les indicateurs.

- **OpenMOLE** [Reuillon et al. 2013] : un moteur de recherche permettant de faire des calculs distribués. Il est utilisé afin d'explorer et calibrer des modèles de simulation. Comme les outils « BehaviourSpace » de NetLogo [Graham 2009] et SimExplorer [Amblard et al. 2003], OpenMOLE permet également de créer facilement et rapidement

des plans d'expériences. Mais en plus de ce que peut offrir ces outils, il permet d'exécuter efficacement en parallèle une partie de l'espace des paramètres d'un modèle grâce à l'utilisation de l'algorithme génétique de type NSGA II (Elitist Non-Dominated Sorting Genetic Algorithm) [Chu et Allstot 2005].

Il serait éventuellement intéressant pour les thématiciens de pouvoir comparer le comportement d'un modèle avec plusieurs exécutions de simulation en parallèle, mais nous ne traiterons pas cette problématique dans le cadre de l'étude.

Par rapport à la problématique considérée, l'effort de ces chercheurs a surtout porté sur l'expression, la génération de plans d'expérience [Goupy 2001] et sur la possibilité d'exécuter des simulations en parallèle ou de les distribuer sur des grappes de machines. Les réalisations informatiques ne traitent pas directement les problèmes d'initialisation et d'observation de modèles de simulation, mais supposent plutôt que les moyens permettant l'initialisation et l'observation des modèles sont déjà mis en place et peuvent être exploités pour réaliser des plans d'expériences (Figure 13). Cependant, nous avons pu identifier certains éléments essentiels afin de pouvoir initialiser et observer les modèles de manière générique.

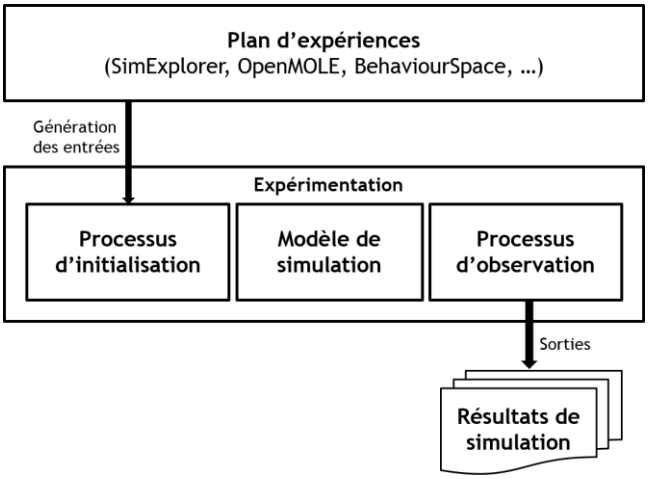


Figure 13. Plan d'expériences versus processus d'initialisation et d'observation de modèle

Les processus d'initialisation et d'observation de modèles de simulation proprement dits sont rarement abordés dans la littérature malgré leur complexité et leur importance dans le processus de modélisation. Nous allons présenter quelques travaux qui ont été menés autour de l'initialisation et de l'observation de modèles afin d'en proposer un processus générique.

3.2 Etat de l'art sur l'initialisation de modèles

Des chercheurs ont essayé de résoudre le problème d'initialisation de modèle en intégrant des techniques et des moyens dans des plateformes de M&S afin de faciliter certaines tâches. D'autres chercheurs ont élaboré des outils plus spécialisés (des plugins ou des frameworks) à l'initialisation et/ou la configuration des modèles. Nous considérons que configurer un modèle fait partie de l'initialisation d'un modèle et qui vise à modifier un certain nombre de valeurs de paramètres et/ou de caractéristiques d'un modèle selon le choix de l'utilisateur.

Nous allons donc présenter quelques-uns d'entre ces outils et ces approches afin de proposer une démarche générique de l'initialisation des modèles.

3.2.1. Initialisation de modèles dans les plateformes de M&S

Les plateformes de M&S intègrent différents techniques et moyens permettant aux modélisateurs et aux informaticiens d'initialiser des modèles de simulation.

- **Initialisation de modèles environnementaux avec TerraME** [Lima et al. 2013]

TerraME est une plateforme de M&S dédiée aux systèmes environnementaux. Il permet de construire des modèles environnementaux à des échelles multiples. Il offre aussi la possibilité d'extraire et d'intégrer des informations à partir des bases de données géographiques avec le langage Terra Modeling Language [Lima et al. 2013]. Cependant, malgré son avantage à représenter facilement les systèmes complexes, les thématiciens (géographes, écologistes, biologistes, économistes, juristes, etc.) ont du mal à l'utiliser à cause de son apprentissage qui nécessite une connaissance en algorithmique et en programmation. Pour résoudre ce problème, le langage TerraME GIMS (Terra ME Graphical Interface for Modeling and Simulation) a été proposé [Lima et al. 2013].Il permet aux utilisateurs de construire des modèles environnementaux graphiquement et de générer automatiquement le code source TerraML correspondant. TerraME GIMS est un environnement de modélisation visuelle pour développer facilement des modèles spatio-temporels dynamiques en utilisant des composants graphiques adaptés aux thématiciens. Il est possible avec cet outil de sélectionner les sources de données dont les thématiciens ont besoin pour initialiser le modèle, en utilisant les composants graphiques fournis par l'éditeur (Properties-View de la Figure 14).

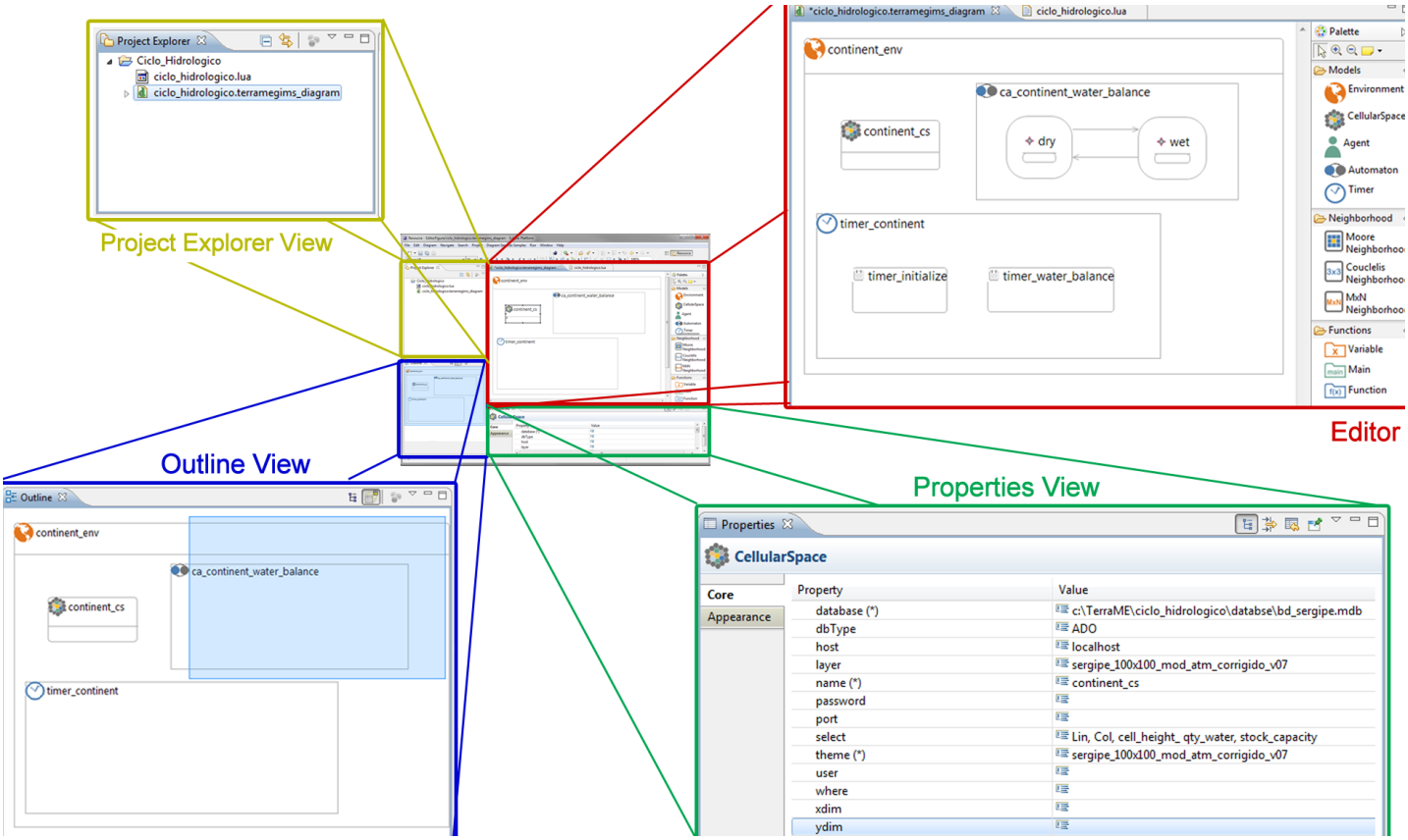


Figure 14. Editeurs du langage TerraME GIMS [Tiago Lima et al. 2013]

Ce langage a été mis en œuvre en utilisant le méta-méta-modèle Ecore du framework EMF, les outils GEF (Graphical Editing Framework) et GMF (Graphical Modeling Framework) pour construire la syntaxe concrète, l'outil EuGENia[10] pour générer facilement des modèles GMF et enfin SWTBot[11] pour améliorer la phase de test dans le processus de développement du modèle.

- **Initialisation de modèle avec Netlogo** [Tisue et al. 2004]

Netlogo est une plateforme de M&S de modèles individus centrés [Tisue et al. 2004]. Elle permet de modéliser facilement des phénomènes complexes comme les domaines de la sociologie, l'écologie, la biologie, l'économie, etc.

Différents moyens sont fournis dans NetLogo pour initialiser un modèle :

  - Configuration du modèle via l'interface graphique de la plateforme, juste en spécifiant les valeurs des variables des composants disponibles (curseurs,

[10]http://www.eclipse.org/epsilon/doc/eugenia/
[11]http://www.eclipse.org/swtbot/

commutateurs, sélecteurs, zones de texte) associés aux agents ou à leurs attributs dans le modèle (Figure 15).

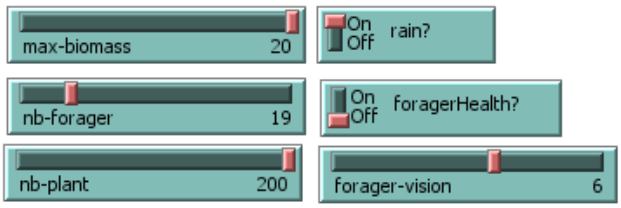


Figure 15. Exemple de composants graphique de Netlogo pour paramétrer un modèle

- Spécification de l'état initial du modèle directement dans le code avec le langage Netlogo en spécifiant les valeurs initiales des différents paramètres (ou variables) associés à chaque type d'agents (Code 1).

```
; initialisation
to setup
  clear-all
  ; environment
  ask patches [ init-patch ]
  ; plants
  set-default-shape plants "vegetation"
  set max-plant-biomass 10
  create-plants 20 [ init-plant]
  ; foragers
  set-default-shape forager "cow"
  set max-forager-energy 20
  create-foragers 50 [init-forager]
  reset-ticks
end
```

Code 1. Initialisation de 20 plantes et de 50 prédateurs à partir du code en Netlogo

- Possibilité de charger et de manipuler des données externes, en particulier les données géographiques raster ou vecteur (avec la fonction « setupGIS »), afin d'initialiser différents éléments (agents) spatialisés du modèle.

- **Initialisation de modèle avec GAMA** [Amouroux et al. 2007]

GAMA (Gis & Agent-based Modelling Architecture) est une plateforme dédiée à la M&S multi-agents. La Figure 16 présente les étapes nécessaires pour mettre en œuvre et explorer un modèle avec GAMA. Pour construire des modèles, elle intègre le langage GAML (GAma Modeling Language) associé à une syntaxe concrète textuelle. En 2013, elle intègre aussi le plugin GAMAGraM [Taillandier 2013] (Figure 17) pour définir plus facilement un modèle à l'aide d'une interface graphique. GAMA est développée depuis 2007 par un groupe actif de chercheurs venant de différentes institutions en France et au Vietnam[12]. Plusieurs projets de recherches ont été déjà

[12]eclipselabs.org/p/gama

développés avec la plateforme GAMA, entre autres, le projet MIRO 2 [Banos et al. 2013], une version plus évoluée de MIRO [Banos et al. 2005] sur la mobilité urbaine, le projet MAELIA [Therond et al. 2014] sur l'impact de la gestion de l'eau sur le système socio-environnemental, etc.

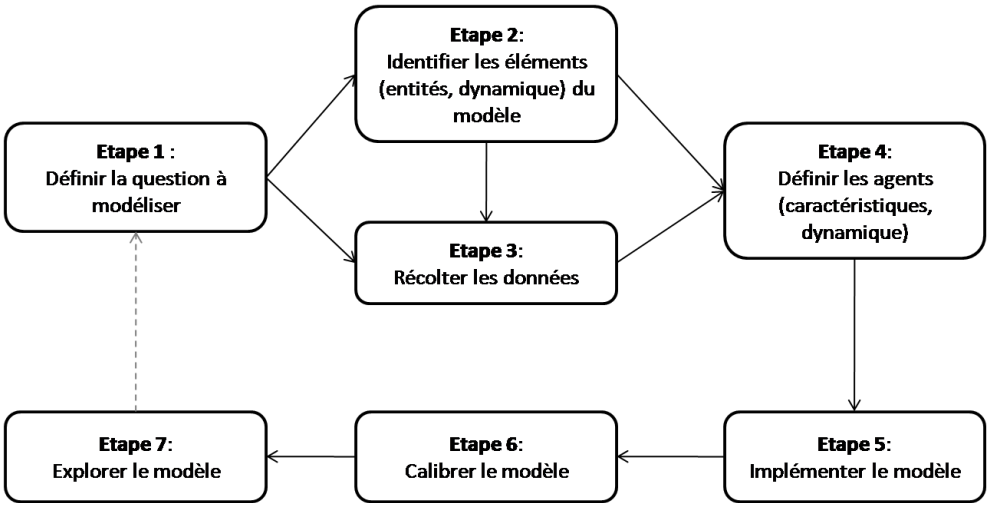


Figure 16. Etapes de modélisation avec GAMA (Figure adaptée d'Amouroux et al. 2007)

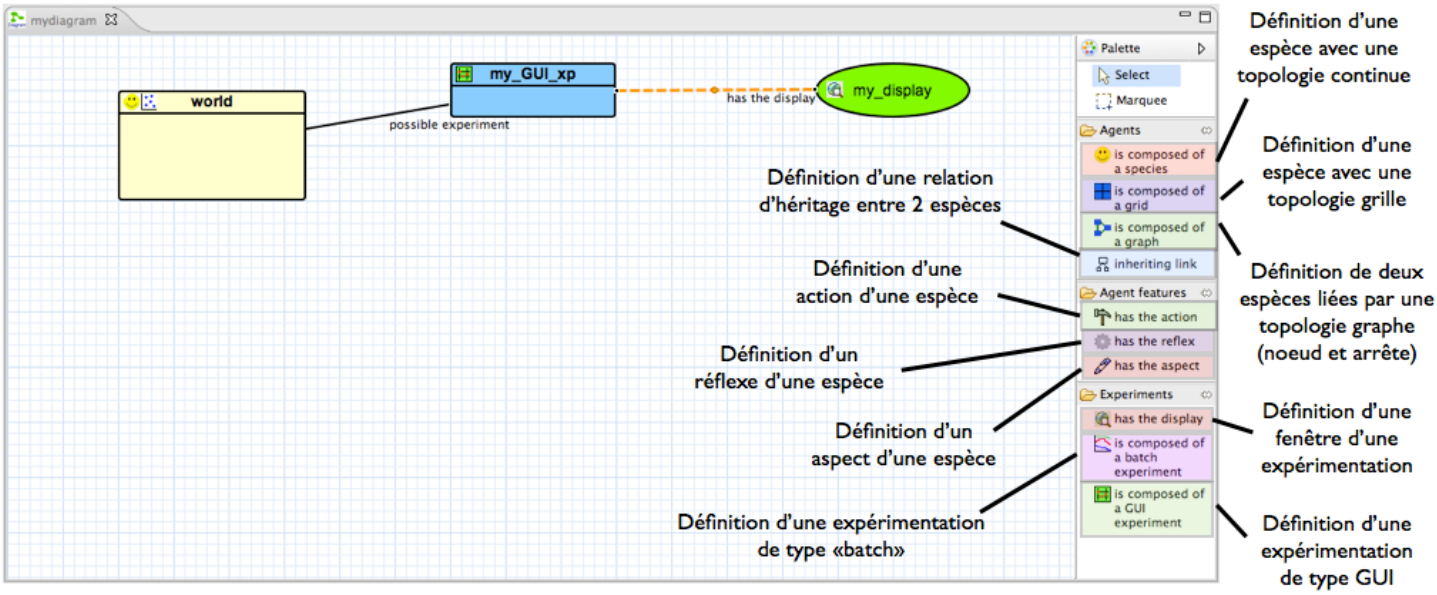


Figure 17. Interface de modélisation de GAMAGraM [Taillandier 2013]

Il existe différentes façons de paramétrer et d'initialiser un modèle dans la plateforme GAMA (Etape 6 de la Figure 16) :

- Configuration du modèle via l'interface graphique de la plateforme, juste en spécifiant les valeurs des variables des composants disponibles (curseurs,

commutateurs, sélecteurs, zone de texte, etc.) associés aux agents ou aux attributs d'agents dans le modèle (Figure 17) ;

- Génération d'un certain nombre d'agents dans le bloc d'initialisation de la simulation du modèle (Code 2) ;

```
model ville

global {
    init {
        create foyer number: 500;
    }
}
```

Code 2. Génération de 500 agents foyer [Banos et al. 2012]

- Spécification et initialisation d'agents à partir d'une source de données externe, à l'aide du mot clé « from », en particulier, les données géographiques (Code 3).

```
file shape_file_batiments <- file("../includes/batiments.shp");
file shape_file_routes <- file("../includes/routes.shp");

create batiment from: shape_file_batiments with: [type:: string(read("NATURE"))];
create route from: shape_file_routes;
```

Code 3. Création d'agents à partir des fichiers shapefiles [Banos et al. 2012]

- Initialisation d'un agent à partir des autres attributs de l'agent (Code 4) ;

```
string type;
int capacite <- type = "Industrial" ? 0 : int(shape.area / 60.0);
```

Code 4. Initialisation de l'attribut capacité en fonction des attributs type et surface [Banos et al. 2012]

Une fois que l'utilisateur lance une expérience, GAMA commence l'initialisation du modèle de simulation en créant tout d'abord l'environnement, puis ensuite les autres agents selon la spécification faite par le modélisateur. Si aucune spécification n'a été faite, alors GAMA va produire une valeur par défaut correspondant au type de chaque attribut d'agent.

- **Initialisation de modèle avec Cormas** [Le Page et al. 2012]

Cormas est un logiciel de simulation multi-agents, développé au sein de l'équipe UPR GREEN du CIRAD. Il permet de modéliser et de simuler les interactions entre les acteurs dans le domaine de la gestion des ressources renouvelables [Le Page et al. 2012]. Cormas permet de développer des

modèles de simulation, en créant des entités avec des attributs, à l'aide de l'éditeur graphique offert par la plateforme ou en utilisant le langage de programmation orienté objet SmallTalk.

Par ailleurs, Cormas offre différentes manières d'initialiser un modèle, à savoir :

- A chaque fois que l'utilisateur crée un nouvel attribut d'une entité, il est possible à partir d'une boîte de dialogue (Figure 18), de spécifier la valeur par défaut de celui-ci ;

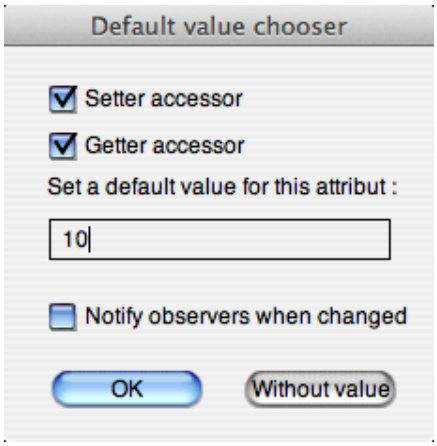


Figure 18. Valeur par défaut d'un attribut [Le Page et al. 2012]

- La possibilité est offerte de changer les valeurs par défaut des attributs de chaque entité, à l'aide d'une interface graphique présentant tous les attributs des entités du modèle (Figure 19).

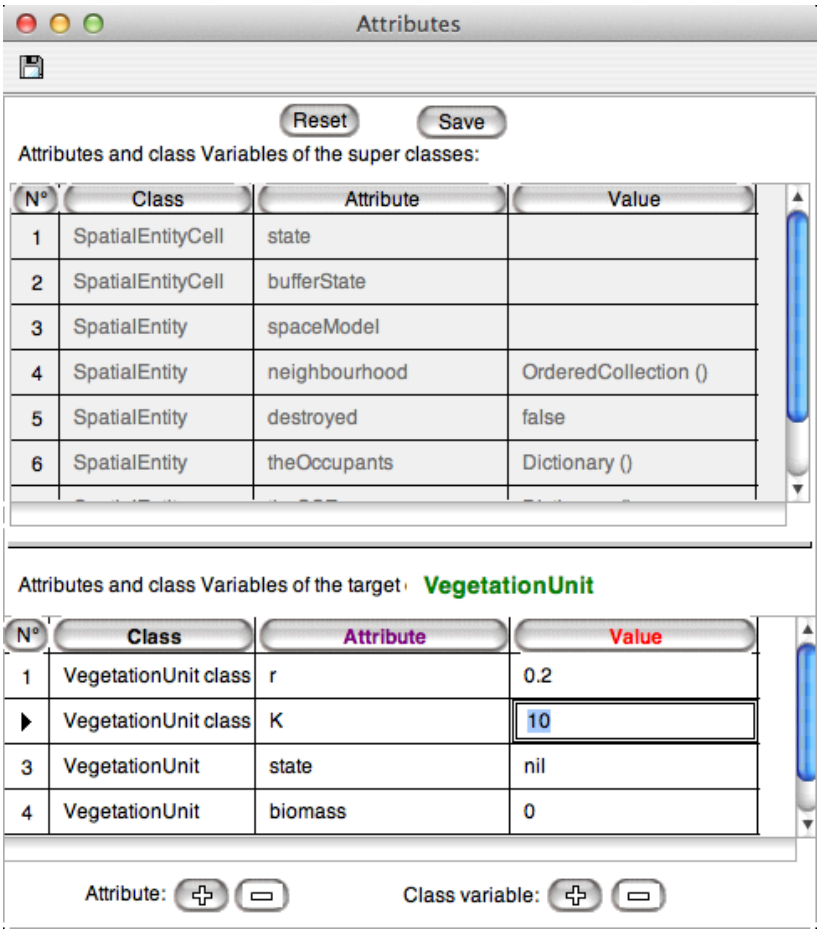


Figure 19. Paramétrage du modèle [Le Page et al. 2012]

- La possibilité est offerte de définir différentes méthodes spécifiques pour initialiser l'environnement du modèle ainsi que chaque attribut des entités de différentes manières. L'utilisateur peut ensuite sélectionner une des méthodes d'initialisation spécifiées avant de lancer la simulation du modèle (Figure 20).

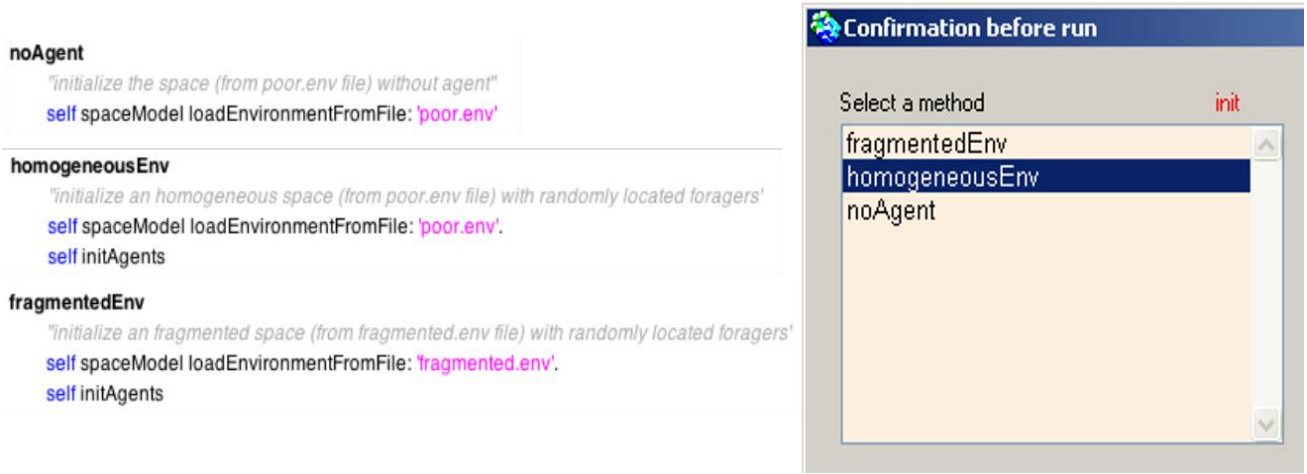


Figure 20. Exemples de méthodes d'initialisation de modèle dans Cormas [Le Page et al. 2012]

- **Initialisation de modèle avec MIMOSA** [Muller 2004]

La plateforme MIMOSA met à disposition des utilisateurs, différents moyens pour pouvoir créer des instances et initialiser le modèle, à savoir :

- La possibilité de spécifier directement les valeurs par défaut des attributs de chaque entité à l'aide d'une boîte de dialogue de l'éditeur graphique (Figure 21);

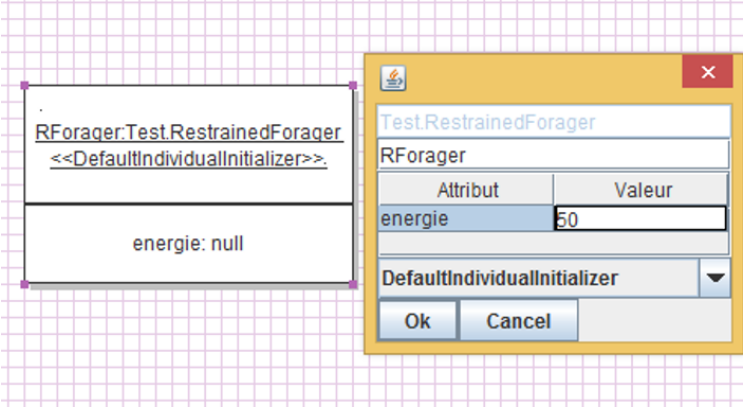


Figure 21. Paramétrage d'une entité dans MIMOSA

- La possibilité de spécifier l'état initial du modèle directement dans le code en Java (via les constructeurs, les accesseurs et les mutateurs ou les variables de classe) pour définir les valeurs initiales des différents paramètres (ou attributs) associés à chaque type d'agents dans le modèle (Code 5).

```
public class LogisticEquation {

    /* Class variables */
    private static final double K = 10000;
    private static final double r = 0.2;

    /**
     *
     */
    public LogisticEquation() {
        // TODO Auto-generated constructor stub
    }
```

Code 5. Exemple d'Initialisation d'une entité en Java

- La possibilité d'initialiser un modèle à partir d'un certain nombre de sources de données externes (actuellement, sous la forme d'un fichier Excel déjà structuré ou d'un fichier shapefile avec leurs données attributaires). Pour cela, MIMOSA offre divers types d'initialiseur permettant de paramétrer et d'initialiser les entités du modèle. Ces initialiseurs peuvent être exploités par l'utilisateur pour spécifier, en code en Java, la façon dont les données seront exploitées pour être conformes aux

structures de données utilisées par le modèle afin de pouvoir initialiser son modèle (Figure 22).

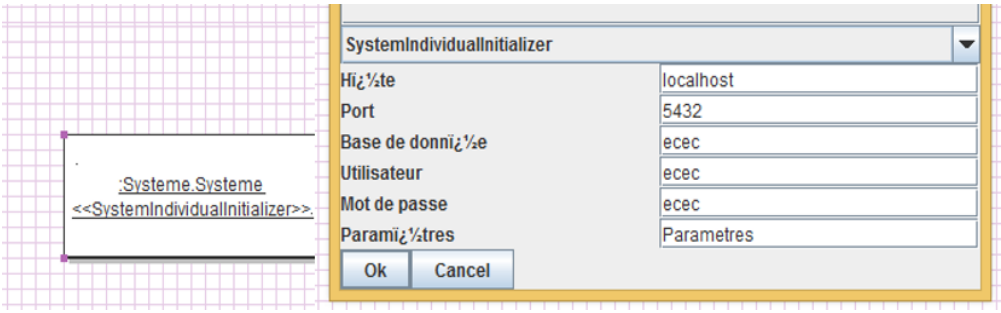


Figure 22. Paramétrage du modèle à partir d'une source de données externe

- **Conclusion**

Ces plateformes de M&S offrent la possibilité d'extraire et d'exploiter des informations à partir de divers supports de données pour faire fonctionner un modèle de simulation. Chacune d'elles dispose d'une interface permettant de saisir directement les valeurs des paramètres du modèle ou de les spécifier explicitement à l'aide d'une syntaxe concrète textuelle (code informatique ou script) ou d'une syntaxe concrète graphique (par exemple, les diagrammes d'objets UML).

Ces solutions proposées sont cependant ad hoc et spécifiques à chaque plateforme. Elles dépendent de la complexité des modèles de simulation implémentés, et changent souvent selon les besoins spécifiques des thématiciens. Pour initialiser les modèles de SES utilisant une grande quantité de données externes, les informaticiens et les modélisateurs sont obligés de travailler sur des codes généralistes comme Smalltalk, Java, C++, etc. et cela nécessite beaucoup de codage de leur part afin de satisfaire les besoins des thématiciens.

Par conséquent, afin d'avoir une interopérabilité entre les langages et les plateformes de M&S, il serait intéressant d'avoir une couche plus abstraite permettant de spécifier de manière générique, l'initialisation des modèles de simulation et de pouvoir générer automatiquement, selon les besoins des modélisateurs et des informaticiens, le code informatique adapté aux langage et plateforme disponibles. Nous allons en décrire quelques tentatives.

3.2.2. Langages dédiés à l'initialisation de modèles

Des langages dédiés à l'initialisation de modèles ont été aussi mis en œuvre par des chercheurs. Entre autres, des langages comme XELOC [David et al. 2009] et MASC [Gaudieux et al. 2014] ont été implémentés pour faciliter l'initialisation des modèles de simulation exploitant des cartes thématiques.

- **Initialisation de SMA avec le langage XELOC** [David et al. 2009]

XELOC (ou eXtensible Editing Language Of Configuration) est un langage de programmation sous forme d'un script basé sur XML. Il a été mis en œuvre afin de faciliter l'initialisation et la configuration des modèles SMA qui ont besoin de beaucoup d'éléments à paramétrer. Il a pour but de faciliter l'initialisation des agents et de leurs environnements en se servant des informations contenues dans des cartes thématiques à la disposition des géographes et de rendre la configuration réutilisable quel que soit le contexte d'implémentation du modèle théorique. XELOC a été appliqué au modèle DS [David et al. 2007]. Pour fonctionner, les modèles comme DS ont besoin de charger des cartes thématiques disponibles auprès des thématiciens (en l'occurrence des géographes), par exemple des cartes de régions, de communes, de densité de population, de répartition des ressources, de mode d'occupation des sols, de potentialité naturelle, agricole, urbaine, etc.

Dans XELOC, une fois le modèle créé, l'utilisateur peut exprimer sous forme de script à base de XML l'initialisation et la configuration de modèle à partir des cartes en possession des thématiciens. L'exécution de ce script va ensuite décrire les traitements nécessaires pour produire l'état initial du modèle.

Il en résulte que, pour initialiser un modèle, XELOC présente comme avantages :

- L'utilisation directe des cartes produites par les géographes;
- Le paramétrage complexe seulement en quelques lignes de code;
- La possibilité de faire un paramétrage conditionnel ou probabiliste afin de produire différentes configurations résultantes dans un intervalle de possibilités bien calibré.

- **Initialisation de SMA avec le langage MASC (MAp Sectors Creator)** [Gaudieux et al. 2014]

Dans les projets SIEGMA [Gaudieux et al. 2014] et EDMMAS [Gangat et al. 2009], les modélisateurs ont ressenti le besoin d'automatiser le processus de découpage de cartes pour gagner du temps et pour pouvoir effectuer facilement des simulations multiples dans un même modèle avec des zones géographiques différentes.

MASC (Map Sector Creator) est une application web facile à utiliser permettant d'automatiser les découpes nécessaires des cartes fournies par les thématiciens. Il génère à partir des cartes, le code d'un langage de programmation directement utilisable dans des plateformes multi-agents tels que Netlogo [Tisue et al. 2004] ou GEAMAS-NG [Conruyt et al. 2009].

MASC est très utilisé dans différents domaines thématiques tels que la géographie, l'économie, la météorologie, l'épidémiologie, etc.

Son objectif est de pouvoir automatiser le processus d'acquisition et de transformation des données cartographiques fournies par les thématiciens sous forme d'image bitmap (au format JPEG, PNG, GIF, TIFF) afin de créer facilement différentes configurations exploitables et compréhensibles directement par le modèle de simulation.

Son utilisation se divise en trois étapes (voir Figure 23) :

- La première étape consiste à introduire toutes les informations nécessaires (taille de cellule souhaitée, liste des couleurs utilisées, etc.) pour pouvoir extraire une coupe de grille de l'espace à partir d'une image bitmap.
- Les données sont ensuite transmises vers le serveur. Le script dans le serveur va manipuler les fichiers bitmaps et extraire une partie de la grille qui va lui servir comme point de départ afin de générer le code utilisé pour l'initialisation du modèle. Ce traitement est effectué automatiquement au niveau du serveur.
- Le résultat produit au niveau du serveur est ensuite transmit de manière graphique au côté client et l'utilisateur obtient un aperçu du résultat du découpage. Il est encore possible à ce niveau de modifier la configuration avant de choisir le type de langage cible pour générer le code final souhaité.

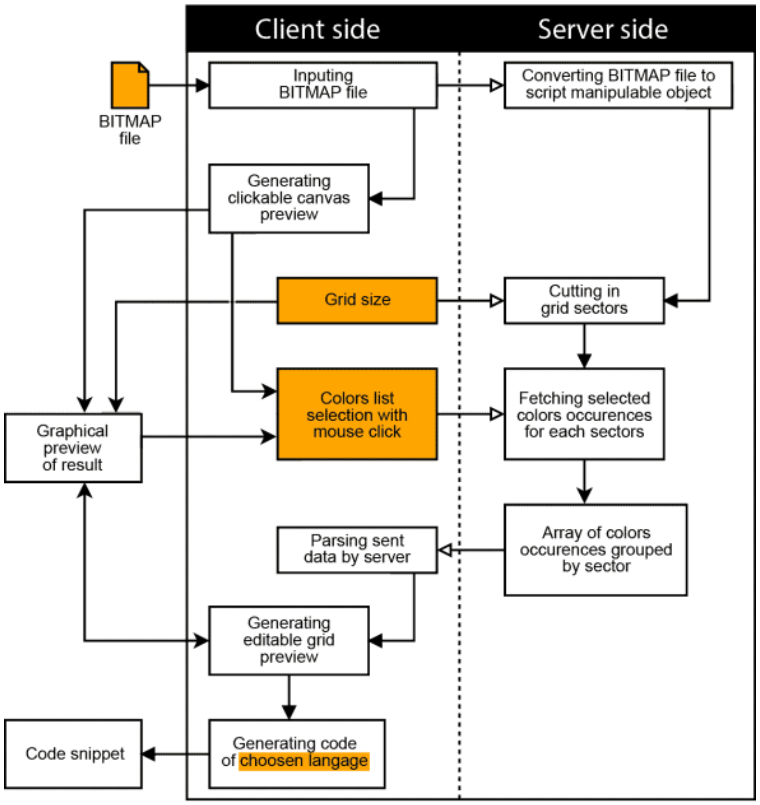


Figure 23. Workflow de MASC [Gaudieux et al. 2014]

MASC permet d'optimiser le temps de travail sur l'initialisation, du fait que plusieurs configurations peuvent être spécifiées en un peu de temps. Son utilisation permet par conséquent, aux modélisateurs et aux thématiciens de se concentrer plus sur les résultats et les observations des simulations.

- **Conclusion**

Les langages XELOC et MASC fournissent les moyens permettant de manipuler des cartes thématiques et d'en extraire les données nécessaires à la configuration et à l'initialisation de modèle de simulation. Cependant, ils ne considèrent que les données cartographiques comme sources de données externes.

3.2.3. Algorithmes pour l'initialisation de modèles

Les informaticiens et les modélisateurs s'intéressent davantage à exploiter ou à construire des algorithmes pour générer automatiquement certains éléments d'un modèle de simulation, afin d'alléger l'initialisation des modèles. Entre autres, des algorithmes sont exploités pour générer par exemple des agents virtuels réalistes dans un modèle de simulation, d'autres encore permettent de paramétrer des agents nouvellement créés de façon automatique.

- **Génération de populations « synthétiques »** [Banos et al. 2013]

Différentes méthodes ont été proposées par les chercheurs dans différents projets mettant en œuvre la génération automatique de populations « synthétiques »[13]. Les méthodes proposées sont basées sur divers algorithmes comme la méthode IPF (Iterative Proportional Fitting) introduite par [Speed 2005], les simulations de Monte-Carlo avec l'utilisation des données d'enquêtes, les probabilités conditionnelles, ou tout simplement la génération aléatoire suivant une loi normale selon un certain nombre de caractéristiques et de comportements des individus, etc.

En particulier, le projet MIRO [Banos et al. 2013] consiste à proposer une méthodologie accompagnée d'un prototype multi-agent individu-centré. Ce projet a pour objectif de représenter, d'analyser, et d'évaluer les dynamiques quotidiennes d'une population urbaine à l'échelle des individus afin de permettre à une ville un développement urbain harmonieux et durable. Un des enjeux du projet est de pouvoir générer dans le modèle SMA, une ville virtuelle avec une population synthétique fiable avec leur emploi du temps respectif qui reflète la réalité, à partir des données disponibles issues des cartes SIG et des enquêtes sur les ménages, y compris leurs déplacements quotidiens (Figure 24).

[13] « Synthétique » : utilisé par Chris Barett [CL. Barrett, et al. 2004] dans le projet TRANSIMS pour parler d'une population de synthèse avec le principe de parcimonie, généré artificiellement par l'utilisateur

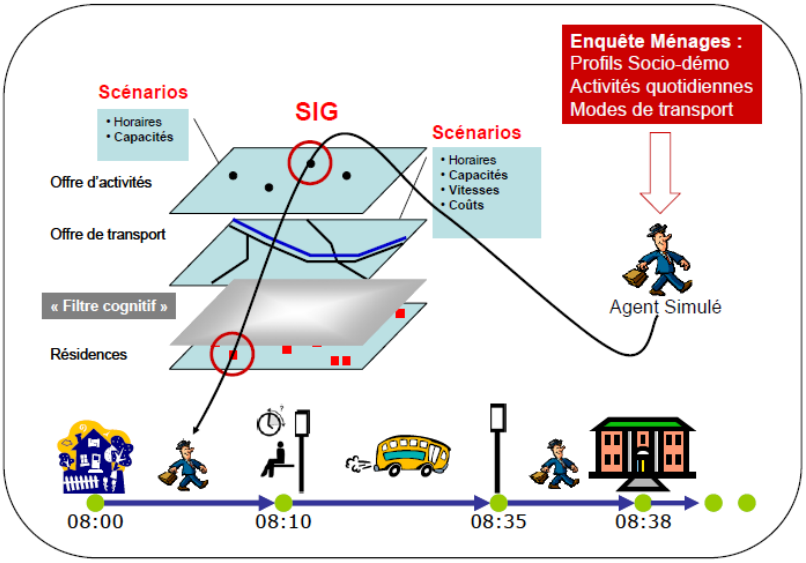


Figure 24. Le cœur du projet MIRO [Banos et al. 2013]

Afin d'initialiser le modèle SMA, trois étapes sont considérées dans le projet:

- Création d'une ville virtuelle avec le plus de détails possible en utilisant des données SIG à la disposition des géographes sur une ville indiquée particulière;
- Création et affectation spatiale d'une population synthétique à partir des données d'enquête sur les caractéristiques des individus et leurs mobilités dans la ville (localisation, activités, mode de transport, temps au domicile, temps au travail, temps de transport, temps de loisir, etc.) ;
- Génération du planning de chaque agent virtuel à partir d'une liste d'activités à accomplir par chaque agent tenant compte des contraintes liées à l'endroit où a lieu l'activité.

Pour générer le planning d'activités de chaque agent, le projet MIRO adopte la démarche modulaire du projet TRANSIMS (TRansportation ANalysis SIMulation System) [Smith et al. 1995] utilisant un synthétiseur de population et un générateur de planning (Figure 25). Sauf qu'au lieu d'utiliser les données SIG et les données de recensements pour générer la population synthétique, le projet MIRO suppose que l'enquête ménage pourrait être exploitée directement (Figure 26) pour générer la population synthétique après avoir appliqué un traitement statistique multivarié (ACM) et les données SIG pour générer le planning d'activités de chaque agent.

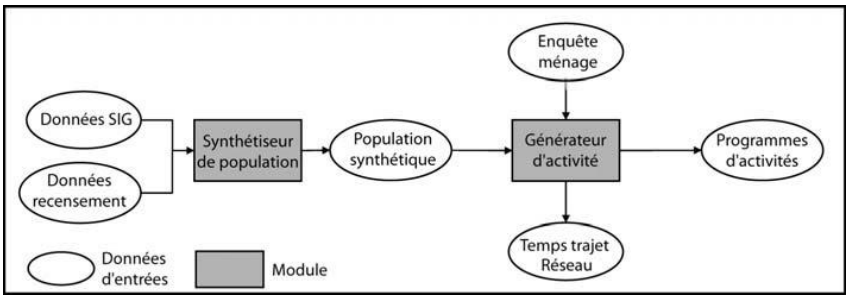


Figure 25. Approche modulaire de TRANSIMS [Banos et al. 2013]

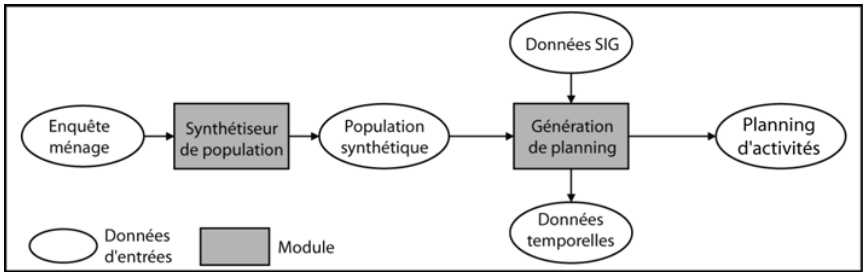


Figure 26. Approche dans MIRO [Banos et al. 2013]

L'approche a été testée dans la communauté d'agglomération de Dijon. L'algorithme de génération de population synthétique d'agents ainsi que leurs emplois du temps respectifs ont été aussi confrontés avec les emplois du temps observés dans l'enquête ménage. Le modèle de simulation reflète plus ou moins la réalité selon la configuration des contraintes horaires liées aux différentes activités.

3.2.4. Synthèse

Des scientifiques ont proposé et implémenté des outils afin d'aborder les problèmes d'initialisation de modèles. En particulier, des plateformes de M&S comme TerraME [Lima et al. 2013], NetLogo [Tisue et Wilensky 2004], GAMA [Amouroux et al. 2007], Cormas [Le Page et al. 2012] et MIMOSA [Muller 2004] mettent à disposition des possibilités pour initialiser un modèle de simulation, soit directement à partir de leur interface graphique respective (ou GUI pour Graphical User Interface), soit à partir des langages associés à des syntaxes concrètes textuelles et/ou graphiques disponibles dans chaque plateforme. Ces langages sont relativement accessibles aux thématiciens. Cependant, ils nécessitent une connaissance en algorithmique et en programmation de leur part. Les langages graphiques comme TerraME GIMS[Lima et al. 2013] intégré dans la plateforme TerraME ou GaMAGraM [Taillandier 2013] intégré dans la plateforme GAMA, sont plus faciles à comprendre et à manipuler que les langages avec des syntaxes concrètes textuelles. Ils permettent de spécifier le paramétrage d'un modèle de manière graphique,

puis de générer le script associé pour pouvoir initialiser un modèle. Ces langages prennent en compte les sources de données externes notamment géographiques afin de faciliter l'initialisation d'un modèle.

D'autres outils plus spécifiques comme XELOC [David et al. 2009] et MASC [Gaudieux et al. 2014] permettent d'automatiser le processus d'acquisition et de traitement de données cartographiques afin de faciliter l'initialisation et la configuration de modèles de simulation à partir des cartes thématiques en possession des thématiciens. Ces outils sont cependant limités à l'exploitation des images cartographiques, et ne considèrent pas d'autres types de sources de données externes.

Des algorithmes sont aussi mis en œuvre par des chercheurs afin de pouvoir générer facilement l'état initial d'un modèle. Par exemple, le cas de la génération de populations « synthétiques » [Banos et al. 2013] qui est basée sur des algorithmes comme IPF (Iterative Proportional Fitting), ou les simulations de Monte-Carlo, ou les probabilités conditionnelles ou tout simplement la génération aléatoire suivant une loi normale.

Nous présentons dans le Tableau 1, quelques caractéristiques de certains outils et plateformes de M&S, par rapport:

- aux données qu'elles peuvent considérer pour alimenter un modèle;
- aux moyens techniques utilisés;
- à la complexité ainsi qu'à la généricité de la méthode d'initialisation proposée dans ces plateformes.

Tableau 1. Quelques plateformes de M&S et outils spécifiques à l'initialisation

Plateforme	Données externes	Langage(s)	Interface graphique	Bloc spécifique	Générateur d'état initial par défaut	Niveau de difficulté	Implémentation spécifique à la plateforme
NetLogo	SGBD spatial	Logo	Oui	Non	Non	Facile	Oui
Cormas	Etat initial existant	Smalltalk	Oui	Oui	Oui	Moyen	Oui
GAMA	SGBD spatial	GAML, GAMAGraM	Oui	Oui	Oui	Facile	Oui
TerraME	SGBD	TML, TerraMEGIMS	Oui	Oui	Non	Moyen	Oui
MIMOSA	SGBD, Excel, Shapefiles	Divers scripts et formalismes	Oui	Oui	Non	Elevé	Oui
XELOC	Carte thématique (Raster)	Script basé sur XML	Non	Non	Non	Moyen	Oui
MASC	Carte thématique (Raster)	-	Oui	Oui	Non	Facile	Oui

A ce propos, par rapport aux problèmes liés à l'initialisation de modèles, nous avons constaté que des chercheurs ont essayé de proposer des outils et/ou des démarches contribuant à faciliter le processus d'initialisation de modèles. Cependant, malgré leurs efforts, les approches proposées sont relativement ad hoc. Elles dépendent de la complexité des modèles implémentés, du formalisme utilisé lors de la modélisation, et elles sont spécifiques à chaque plateforme.

Plus les structures de données sont nombreuses et hétérogènes, plus l'initialisation nécessite beaucoup de codage et des fois, on est même obligé de travailler sur des codes généralistes comme du Java ou du C++ pour répondre à certains besoins.

3.3 Etat de l'art sur l'observation de modèles

Dans cette section, nous allons examiner quelques approches que les outils de M&S mettent en œuvre pour aider les utilisateurs à explorer les modèles de simulation. L'objectif est de pouvoir détecter les éléments nécessaires à l'observation afin d'élaborer un processus d'observation générique, indépendamment de la technologie, des langages ou des plateformes de M&S utilisés.

3.3.1. Observation de modèles dans les plateformes de M&S

Les plateformes de M&S offrent des moyens permettant aux modélisateurs et aux thématiciens d'observer ce qui se passe dans les modèles de simulation.

- **Observation de modèles avec Netlogo** [Tisue et Wilensky 2004]

La plateforme NetLogo fournit des moyens permettant de spécifier les sorties ainsi que leurs présentations visuelles. D'abord, l'utilisateur peut directement utiliser le langage NetLogo pour calculer les sorties dont il a besoin pendant la simulation. Ensuite, afin de les visualiser à l'écran, NetLogo met à disposition des utilisateurs, des composants graphiques tels que les histogrammes, les courbes, les zones de texte, etc. (Figure 27). Ces composants sont paramétrables et peuvent afficher l'évolution des sorties calculées à partir du code.

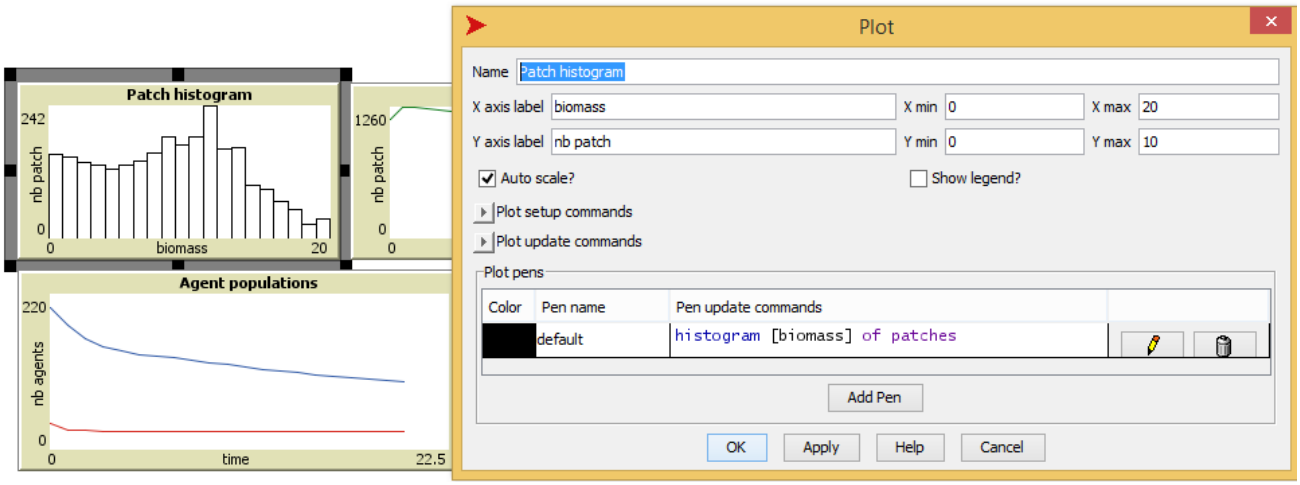


Figure 27. Paramétrage d'un composant graphique dans NetLogo

Par rapport au processus d'observation de modèles, d'une part, la simulation de modèles développée en NetLogo est gérée par un compteur à pas de temps constant donc il ne permet pas de définir différents types de stratégies d'observation. Sur un autre plan, la structure d'un code

NetLogo ne permet pas au modélisateur de distinguer facilement les instructions propres à l'observation de modèle car toutes les instructions sont mélangées dans un bloc de code. Par conséquent, lorsque les besoins en termes d'observation changent, tout le code source qui implémente le modèle de simulation est affecté par ce changement.

- **Observation de modèles avec GAMA** [Amouroux et al. 2007]

La plateforme GAMA permet aussi d'explorer ce qui se passe pendant la simulation du modèle (step 7 de la Figure 16). Le bloc d'instruction « experiment » (Code 6) permet de spécifier les entrées du modèle et les sorties que les modélisateurs ou les thématiciens souhaitent observer pendant la simulation. Il permet aussi la présentation visuelle (image, graphique, texte, fenêtre de visualisation, moniteurs, etc.) selon le type d'aspect défini par l'utilisateur. Définir l'aspect d'un objet (ou d'un agent) revient à décrire comment l'objet (ou l'agent) sera présenté visuellement/graphiquement dans le modèle. Dans GAMA, il est possible de définir les différents aspects pour un même agent et de le modifier au cours de la simulation.

```
experiment my_model type : gui {
    /** Définition des entrées et des sorties du
    modèle
    */
}
```

Code 6. Bloc d'instruction dédié à l'expérimentation d'un modèle dans GAMA

Par rapport au processus d'observation de modèles, GAMA fournit un bloc d'instruction dédié à la spécification des entrées et des sorties du modèle de simulation, ce qui permet déjà d'améliorer la visibilité des préoccupations dans le processus de modélisation. De plus, la notion d'aspect utilisée par la plateforme GAMA est intéressante, et pourrait être exploitée car elle permet de répondre aux besoins multiples des thématiciens en termes de visualisation. Cependant, il serait encore mieux à notre avis de pouvoir distinguer les spécifications liées au processus d'observation avec celles de l'initialisation de modèles même si les sorties dépendent des entrées afin de réduire la complexité.

- **Observation de modèles avec Cormas** [Le Page et al. 2012]

La plateforme Cormas permet aussi de spécifier l'observation d'un modèle de simulation. Il est possible de calculer à partir d'un code en Smalltalk, les sorties du modèle puis de spécifier des

sondes (ou probes en anglais) afin de pouvoir suivre l'évolution des éléments du modèle que l'on souhaite observer pendant la simulation. Ce processus est représenté sur la Figure 28.

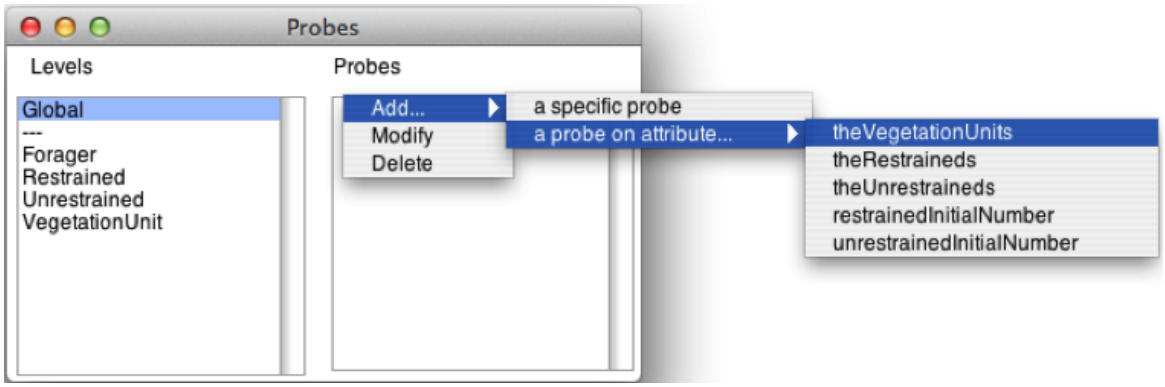


Figure 28. Spécification de sondes pour la visualisation du modèle dans Cormas [Le Page et al. 2012]

D'autre part, il fournit le moyen à l'utilisateur de créer des méthodes de type « point of view » (PoV) (ou point de vue) afin de définir comment chaque élément du modèle sera présenté à l'écran (sa forme, sa couleur, sa taille, son emplacement, etc.) (Figure 29).

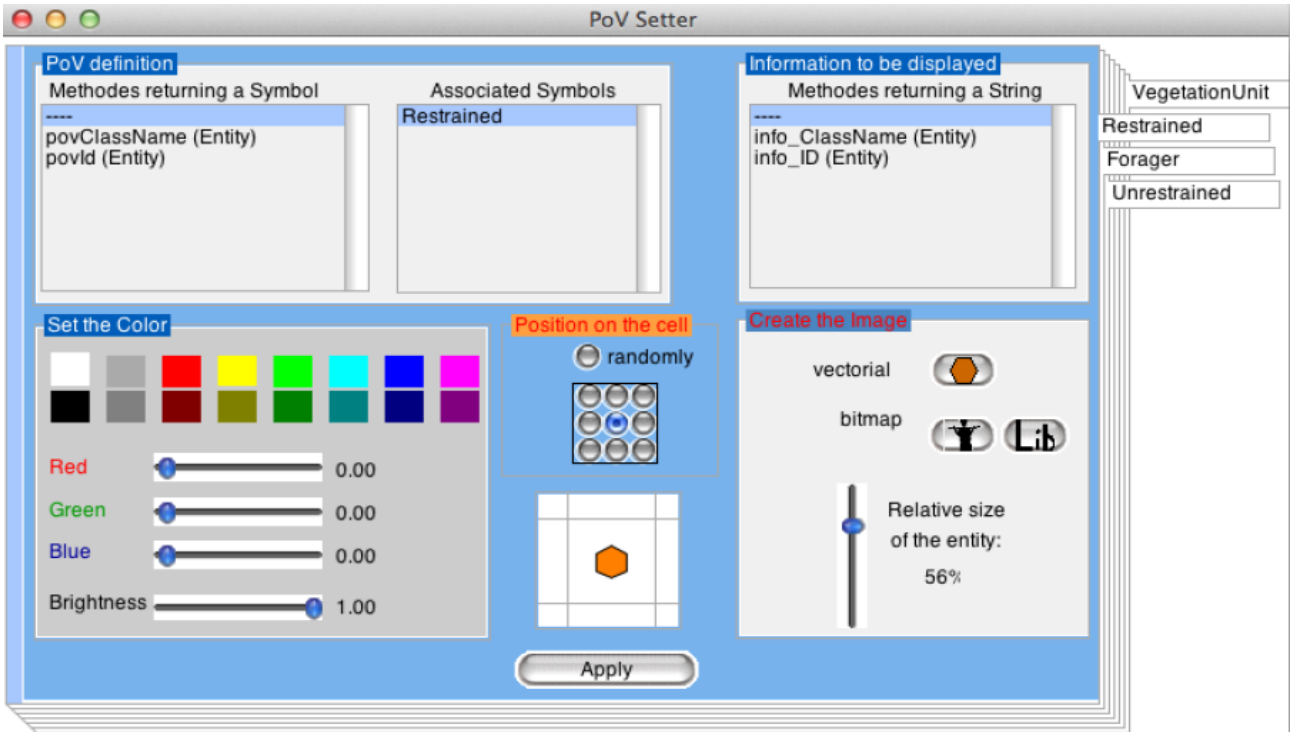


Figure 29. Spécification d'un point de vue d'une entité dans Cormas [Le Page et al. 2012]

L'utilisation de la plateforme Cormas met en évidence deux grandes étapes de spécification pour observer un modèle de simulation, à savoir :

- La spécification d'un observateur à l'aide des sondes (ou probes) permettant de récupérer l'état des éléments observables pendant la simulation en cas de changement d'état ;
- Celle de la présentation visuelle des éléments au cas où l'utilisateur souhaite les visualiser.

- **Observation de modèles avec MIMOSA** [Müller 2004]

Dans MIMOSA, il est aussi possible de spécifier l'observation d'un modèle de simulation. Elle consiste à définir :

- ***Des observations par notification*** : Elles sont spécifiées à travers des sondes (ou probe en anglais). Ces sondes permettent d'extraire l'état des éléments observables du modèle de simulation suivant une stratégie d'observation définie dans le code. Par exemple, le bout de code présenté par le Code 7 montre les instructions permettant de spécifier une sonde qui va permettre de suivre l'évolution de la quantité de biomasse sur une parcelle donnée.

```
for (Resource resource:holder.getResources()){
    state.sendProbe("biomass",
            new Object[]{holder, resource, holder.getQuantity(resource)});
}
```

Code 7. Exemple de spécification d'une sonde dans MIMOSA

- ***Des observations par échantillonnage utilisant la notification*** : Il est possible dans MIMOSA de spécifier à quel moment de la simulation on voudrait récupérer l'état du modèle de simulation.

- ***Les sorties*** : Les sondes peuvent être envoyées sur plusieurs supports de sorties. Ces derniers peuvent être des composants graphiques comme des grilles, des courbes, des graphes ou des supports de données comme des bases de données, des fichiers, des outils statistiques, etc. éventuellement utilisés pour des analyses de données plus approfondies. Par exemple, le bout de code présenté par le Code 8 permet de stocker l'évolution de la quantité de biomasse sondée par la sonde, dans une table d'une base de données. Il est aussi possible pour l'utilisateur de spécifier directement les sorties à l'aide des composants graphiques fournis par l'outil représentées sur la Figure 30, si les sondes et le type de sortie ont été spécifiés.

```
        stmt = dbConn.createStatement();
        stmt.executeUpdate("CREATE TABLE \"" + spec.getBiomassTable()
                        + "\" (TIME Integer NOT NULL,OBJECT varchar(20), RESOURCE varchar(20), QUANTITY numeric)");
        stmt.close();

        // prepare statements
        addBiomass = dbConn.prepareStatement("INSERT INTO \"" + spec.getBiomassTable() + "\" VALUES (?,?,?,?)");
        Mimosa.addCloseable(this);
    }
    if (probe.getName().equals("biomass")) {
        log.debug("Output2DataBase: getting biomass description.");
        addBiomass.setLong(1, probe.getGlobalTime());
        addBiomass.setString(2, probe.getString(0).substring(0,Math.min(probe.getString(0).length(),20)));
        addBiomass.setString(3, probe.getString(1).substring(0,Math.min(probe.getString(1).length(),20)));
        addBiomass.setDouble(4, probe.getDouble(2));
        addBiomass.executeUpdate();
    }
```

Code 8. Exemple de spécification de sortie en utilisant le code en Java dans MIMOSA

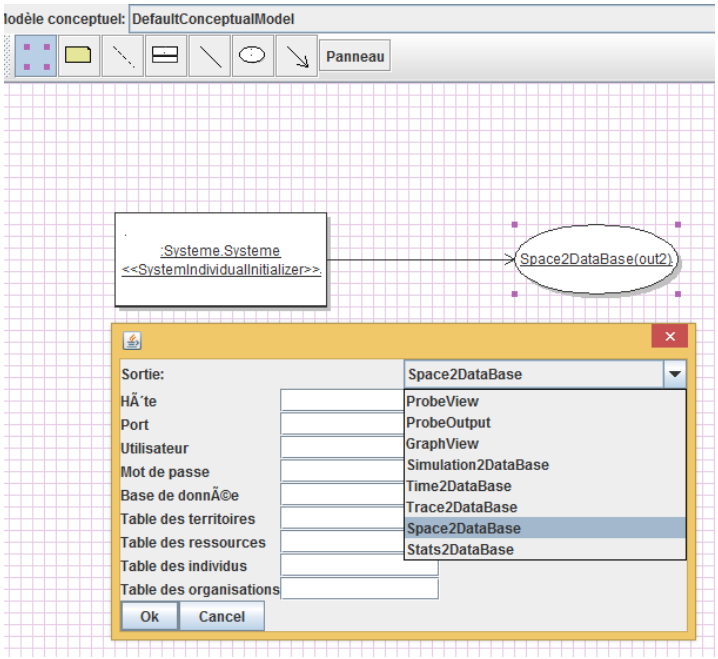


Figure 30. Exemple de spécification de sortie en utilisant l'interface graphique de MIMOSA

Ainsi, à part la possibilité de spécifier des éléments observables avec leurs présentations visuelles, MIMOSA offre aussi un moyen pour définir des stratégies d'observation ; ce qui répond aux besoins des thématiciens. Malgré cela, implémenter une observation dans MIMOSA demande une compétence très avancée en algorithmique et en programmation et nécessite beaucoup de codage de la part des utilisateurs. Il est donc intéressant de mettre en œuvre des outils à la portée des thématiciens et des modélisateurs qui leur permettent de spécifier ces étapes de manière plus simplifiée, puis de générer le code informatique éventuellement complexe permettant d'observer les indicateurs qu'ils souhaitent observer.

- **Mécanisme d'observation issu du formalisme DEVS**

DEVS tient une place importante dans la théorie de M&S et il est considéré actuellement comme le dénominateur commun des systèmes de modélisation multi-formalisme [Vangheluwe 2000]. Les dynamiques des modèles multi-agents sont, en général, basées sur le formalisme DEVS [Zeigler et al. 2000]. Le comportement d'un élément d'un système est capturé par un bloc appelé « DEVS atomique » qui est basé sur un ensemble M de 7-uplet, tel que $M = < X, Y, S, \delta_{ext}, \delta_{int}, \lambda, t_a >$ (Figure 31), avec :

- X : l'ensemble des évènements en entrée ;
- Y : l'ensemble des évènements en sortie ;
- S : l'ensemble des états possibles ;
- $\delta_{ext} = X\ x\ Q \rightarrow S, o\grave{u}\ Q = S\ x\ dur\acute{e}e$: la fonction de transition externe ;
- $\delta_{int} = S \rightarrow S$: la fonction de transition interne ;
- $\lambda = S \rightarrow Y$: la fonction de sortie ;
- $t_a = S \rightarrow dur\acute{e}e$: la fonction d'avancement du temps.

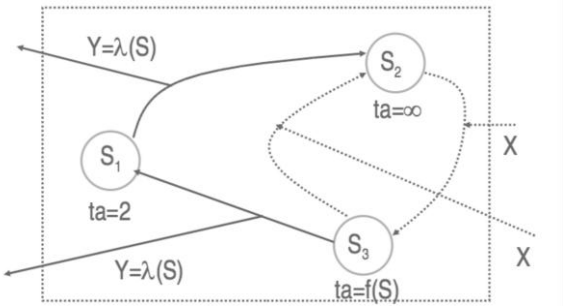


Figure 31. DEVS atomique [Müller 2007]

Quesnel G. a proposé un mécanisme d'observation [Quesnel et al. 2012], qui permet d'observer des modèles hétérogènes, et qui n'influence pas le comportement normal des modèles pendant la simulation. Le principe est d'étendre DEVS-Parallèle [Chow et al. 1994] à de nouveaux ensembles de ports d'entrée et de sortie et à une nouvelle fonction de transition. D'où, nous avons le t-uplet $M' = < X, Y, \boldsymbol{Y_{obs}}, S, \dots, \delta_{ext}, \delta_{int}, \lambda, \boldsymbol{\lambda_{obs}}, t_a >$, tel que $\lambda_{obs} : S \rightarrow Y_{obs}$

Deux modes d'observation sont pris en compte par ces extensions, à savoir :

- Le mode évènementiel : un évènement est créé par la fonction λ_{obs} à chaque changement d'état ;
- Le mode planifié : chaque pas de temps du modèle génère un évènement Y_{obs} ;

Cette extension de DEVS-Parallèle est implémentée dans la plateforme VLE[14] (Virtual Laboratory Environment) [Quesnel et al. 2012], et est appliquée à la plateforme RECORD[15] qui est une plateforme de M&S informatique dédiée à l'étude des agro écosystèmes.

3.3.2. Technique liée à l'observation des modèles de simulation

- **Modèle de conception « Observateur »**

Dans le monde de la programmation, notamment, l'orienté objet, de plus en plus de modèles de conception (ou design pattern) sont élaborés afin de gagner plus de temps et faciliter le développement des logiciels » [Gamma et al. 1995]. Ces modèles de conception décrivent des solutions abstraites, indépendamment d'un langage particulier, pour résoudre des problèmes répétitifs que les programmeurs rencontrent la plupart du temps dans la conception et l'implémentation des logiciels afin de permettre aux informaticiens de ne pas réinventer la roue. Ils se divisent en trois grandes catégories, à savoir : le modèle de *création*, celui de *structure* et celui de *comportement*. Le modèle de conception « Observateur » fait partie de celui de la troisième catégorie.

Par définition, le modèle de conception « *observateur »* définit une relation entre les objets de type un à plusieurs, de façon à ce que, lorsqu'un objet change d'état, tous ceux qui en dépendent en soient informés et soient mis à jour automatiquement [Mohamed 2014]. Ce modèle a pour but de fournir au développeur un moyen simple permettant à un objet de tenir informé d'autres objets, qui lui sont abonnés, de son changement d'état.

Le diagramme de classe simplifié représenté sur la Figure 32 présente globalement le principe du modèle de conception « observateur ».

[14]VLE est un environnement complet de multi-modélisation et de simulation de système complexe. Il est composé d'une bibliothèque de modélisation DEVS C++ avec des extensions (équation différentielle, équations aux différences, réseaux de Pétri, automates cellulaires, équations différentielles spatialisées), de simulateur DEVS, de GUI de modélisation, de GUI de visualisation, d'un paquet R etd'un paquet Python)

[15]RECORD : REnovation et COoRDination de la modélisation des cultures pour la gestion des agroécosystèmes (http://www.vle-project.org/wiki/Projects_Using_VLE)

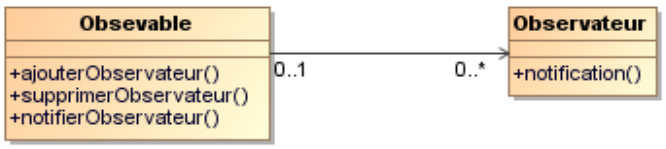


Figure 32. Modèle de conception "Observateur"

La catégorie « Observable » représente l'objet observé et celle de l'« Observateur » représente les objets abonnés qui seront informés de l'évolution de l'état de l'objet observé. Afin d'envoyer un signal aux objets abonnés lors d'un changement d'état (avec la méthode *notifierObservateurs()*), l'objet observé doit intégrer la liste contenant ses abonnés. Pour cela, les méthodes *ajouterObservateur()* et *supprimerObservateur()* lui permettent, respectivement, d'ajouter et de supprimer des observateurs dans sa liste des abonnés. Les objets observateurs n'implémentent que la méthode *notification()*. Cette dernière permet aux observateurs d'être notifiés lors d'un changement d'état de l'objet observé.

3.3.3. Synthèse

A part la capacité de créer et d'initialiser des modèles de simulation, les outils de M&S comme Netlogo, GAMA, Cormas, MIMOSA, etc. disposent de plug-ins permettant d'explorer des modèles de simulation afin d'aider les modélisateurs et les thématiciens à observer ce qui se passe pendant la simulation, à analyser et à comprendre leur comportement. Par ailleurs, des techniques permettant l'observation et le suivi de ce qui se passe dans le modèle pendant la simulation sont aussi élaborées. Certaines plateformes implémentent par exemple le modèle de conception « Observateur » [Gamma et al. 1995] et d'autres exploitent d'autres approches comme le mécanisme d'observation qui étend le formalisme DEVS-Parallèle [Quesnel et al. 2012].

Le Tableau 2 présente quelques caractéristiques de certaines plateformes, par rapport à l'observation de modèles.

Tableau 2. Quelques plateformes de M&S avec leurs propriétés d'observation

Plateforme	Spécification des sorties	Spécification de la présentation visuelle	Interface graphique	Bloc spécifique	Stratégie d'observation	Implémentation spécifique à la plateforme
NetLogo	Oui	Oui	Oui	Non	Pas de temps constant	Oui
Cormas	Oui	Oui	Oui	Oui	Sonde en cas des changement d'état (Pattern Observateur)	Oui
GAMA	Oui	Oui	Oui	Oui	Pas de temps constant	Oui
TerraME	Oui	Oui	Oui	Oui	Pattern Observateur	Oui
MIMOSA	Oui	Oui	Oui	Oui	Par observation (sonde et pas de temps constant)	Oui

Comme à l'initialisation, en dépit des travaux menés par les chercheurs, spécifier une observation à partir des plateformes de M&S nécessite beaucoup de codage et de compétence en algorithmique et en programmation et les mécanismes d'observation de modèles existants sont relativement ad hoc. Ils dépendent du formalisme utilisé et sont spécifiques à chaque plateforme. Aussi, nous proposons de mettre en œuvre des outils permettant aux thématiciens et aux modélisateurs de spécifier facilement les éléments nécessaires au processus d'observation de modèles et c'est à charge par ces outils de générer automatiquement le code informatique permettant de le réaliser. Inspirer des techniques associées à l'observation de modèles, nous proposons de mettre en œuvre, deux types de stratégie d'observation de modèles, à savoir : une première stratégie permettant au modèle de signaler l'observateur lors d'un changement d'état éventuel et une autre permettant à l'observateur d'observer l'état du modèle à partir des dates d'observation définies par l'utilisateur.

3.4 Conclusion

Nous avons vu au début de cet état de l'art que les outils comme BehaviorSpace, SimExplorer et OpenMOLE se concentrent plus sur la gestion d'un ensemble d'expérimentations de modèles et non sur la spécification du processus d'initialisation et d'observation proprement dit.

Face aux problèmes liés à l'initialisation et à l'observation des modèles de simulation, des chercheurs ont essayé de proposer des outils et/ou des démarches contribuant à faciliter le processus de l'initialisation et de l'observation de modèles ou de systèmes en général. Malgré leurs efforts, les approches proposées dans la littérature sont relativement ad hoc et limitées. Elles

dépendent du formalisme utilisé lors de la modélisation, et elles sont spécifiques à des modèles particuliers. Aucun cadre général de processus d'initialisation et d'observation de modèle n'a encore été envisagé.

Proposer une démarche générique des processus d'initialisation et d'observation de modèle de simulation, inspirée des différentes approches, nous semble essentiel. Celle-ci doit permettre d'une part, d'exploiter les données ainsi que les informations issues de diverses sources de données hétérogènes que les thématiciens possèdent afin de pouvoir initialiser les modèles ; d'autre part, d'observer des sous-parties et des sous-parties agrégées du modèle et de prendre en compte les différentes stratégies d'observation possibles afin de pouvoir générer facilement les indicateurs souhaités par les thématiciens lors de la simulation.

Chapitre IV. Méthodologie proposée

4.1. Introduction

Cette étude vise à proposer un cadre générique de spécification de l'initialisation et de l'observation de modèles de SES en mettant à disposition des thématiciens et des modélisateurs, les concepts pertinents indépendamment de leur réalisation technique, séparant ainsi le « quoi » (qu'est-ce que l'initialisation et l'observation ?) du « comment » (réalisation sur une plateforme de simulation particulière). Pour cela, d'une part, le processus d'initialisation de modèles revient à décrire les sources de données hétérogènes à la disposition des thématiciens et leurs contenus, à décrire le modèle de simulation indépendamment des structures de données sous-jacents, que nous considérons comme étant des fonctions de temps $\frac{dx}{dt} = f(x)$, puis à décrire la chaîne de transformations possibles entre ces spécifications afin de générer du code manipulant les structures de données et permettant de construire l'état initial du modèle avec les paramètres des fonctions f .D'autre part, le processus d'observation revient à décrire les structures du modèle qu'on a besoin d'observer, ensuite à définir la stratégie d'observation engendrant des trajectoires, et finalement à spécifier une chaîne de transformations de ces trajectoires afin de générer du code permettant de construire les indicateurs souhaités par les thématiciens.

Dans la littérature, une nouvelle pratique de l'ingénierie des systèmes appelée IDM (ou Ingénierie Dirigée par les Modèles), exploitant les dernières technologies informatiques et fournissant des moyens et des outils dédiés à la méta-modélisation, est de plus en plus utilisée pour aborder efficacement d'une part, les problèmes de compatibilité entre les outils et les langages informatiques dus à l'évolution de la taille et de la complexité des logiciels dans les industries de génie logiciel et d'autre part, les problèmes de mise en correspondance, de tissage et de transformation entre modèles ou les structures de données hétérogènes créées par différents modélisateurs. L'IDM permet également de mettre en œuvre des langages adaptés à une application spécifique (ou DSL pour Domain Specific Language). Ces DSL permettent aux experts de domaines (éventuellement, non informaticiens) de construire facilement des modèles tout en restant sur leurs préoccupations.

Implémenter des DSL permet ainsi de fournir des outils dédiés aux thématiciens afin de rendre facile le processus d'initialisation et d'observation de modèles de simulation.

Dans ce chapitre, nous présenterons succinctement les principaux concepts d'IDM, les outils associés pour justifier le choix de la méthodologie adoptée et la mise en œuvre de DSL afin de résoudre les problèmes de l'initialisation et de l'observation de modèles.

4.2. Ingénierie Dirigée par les Modèles (ou IDM)

4.2.1. Présentation synthétique de l'IDM

Selon Jézéquel, l'IDM(en anglais, MDE pour Model Driven Engineering), est une nouvelle méthodologie de conception et de développement de logiciels centrée sur les modèles [Jézéquel et al. 2012].Afin de ne pas confondre la notion de « modèle »au sens de l'IDM avec le « modèle de simulation » qui est une implémentation informatique permettant de représenter et de simuler un système du monde réel comme les SES, dans la suite de ce rapport, nous utiliserons le mot « modèle » tout court, pour parler d'un modèle en IDM et « modèle de simulation » ou « modèle » suivi de son nom, pour parler des modèles de SES comme le modèle Mirana [Aubert et al. 2010]. Bien qu'un modèle en général, selon Minsky, soit défini comme étant une représentation abstraite d'un système selon un certain point de vue pour répondre à une question sur celui-ci [Minsky 1965] ; en IDM, le souhait est surtout de pouvoir décomposer les différentes préoccupations pertinentes d'une application logicielle sous forme de modèles et de pouvoir ensuite les mettre en correspondance afin d'aborder efficacement la construction d'une application. Par exemple, d'une part, pour aborder le problème d'initialisation des modèles de simulation, que nous considérons comme notre première préoccupation dans le processus de modélisation, nous pouvons identifier comme modèles : la description des sources de données hétérogènes, celle des structures de données d'entrée du modèle de simulation, celle des structures de données du modèle de simulation, et enfin celle de la chaîne de transformations entre les deux structures de données ; d'autre part pour le problème d'observation des modèles de simulation, qui est notre deuxième préoccupation, nous pouvons identifier comme modèles : la description d'une partie du modèle de simulation que les thématiciens veulent observer et suivre, celle des indicateurs qu'ils souhaitent construire et celle des transformations permettant d'obtenir et de présenter les indicateurs à partir des parties observables du modèle de simulation.

En effet, IDM est une forme d'ingénierie générative [Combemale 2008] dans laquelle tout ou partie d'une application est engendrée à partir de modèles. Elle offre des moyens pour construire facilement des DSL. Par définition, un DSL est un langage de programmation (ou de

modélisation) dont les spécifications sont dédiées à un domaine d'application précis comme par exemple : les logiciels en temps réel, les jeux vidéo, les langages comme HTML qui est un DSL dédié à l'écriture de documents contenant des liens hypertextes, PHP qui est un DSL dédié à la production des pages web et à la gestion des bases de données, SQL qui est un DSL pour interroger ou pour manipuler les bases de données relationnelles, et même Fortran, qui est le premier langage de programmation, peut être en ce moment considéré comme étant un DSL dédié au calcul scientifique. Actuellement, les DSL sont de plus en plus utilisés pour développer des applications complexes [Volter et al. 2013]. Un des avantages de son utilisation est de permettre aux thématiciens et aux modélisateurs de manipuler directement les concepts spécifiques à leur domaine d'application, de travailler à un niveau plus abstrait afin de se concentrer sur leurs préoccupations ainsi qu'à leur domaine de recherche, puis de générer facilement des modèles ou du code informatique reflétant la complexité des explications qu'ils se donnent de la réalité. Pour faciliter la construction des DSL, l'Object Management Group a proposé la pyramide de modélisation (Figure 33) qui s'articule autour de cinq concepts fondamentaux: *le système réel* que les thématiciens veulent modéliser, *le modèle* qui permet de représenter ce système, *le méta-modèle* qui permet de définir les concepts du modèle, *le méta-méta-modèle* qui sert à spécifier le méta-modèle et *les transformations* entre les différents niveaux de la pyramide [El Hamlaoui 2012].

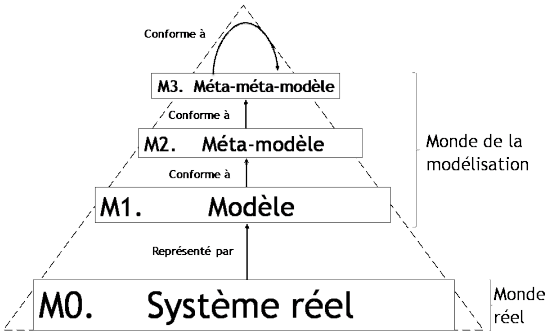


Figure 33. Pyramide de modélisation de l'OMG [Figure adaptée de l'OMG 2014]

4.2.1.1. Système

C'est le monde réel ou le domaine de discours des thématiciens qui est représenté à la base de la pyramide de modélisation (niveau M0). Un système socio-écologique est difficilement expérimentable en général, donc il est nécessaire d'avoir sa représentation simplifiée en respectant

les principes de parcimonie[16] et de substituabilité[17]et il faut que cette représentation soit manipulable afin de pouvoir répondre à des questions que l'on se pose sur le système.

4.2.1.2. Modèle

Nombreuses sont les définitions, mais nous retenons la définition de Minsky en 1965, « A est un modèle d'un système B pour un observateur X, si manipuler A permet de répondre à des questions de X sur le système B ». Bézivin rejoint cette définition et présente un modèle (niveau M1) comme étant une abstraction simplifiée d'un système étudié, construite dans une intention particulière et qui doit pouvoir être utilisée pour répondre à des questions sur le système [Bézivin 2003].

4.2.1.3. Méta-modèle

Un méta-modèle est un modèle qui définit le langage d'expression d'un modèle, i.e. le DSL. Il définit les concepts ainsi que les relations entre concepts nécessaires à l'expression des modèles. Un modèle est dit « conforme à » un méta-modèle d'après [Jézéquel et al. 2012] si chacun de ses éléments est instance d'un élément du méta-modèle et s'il respecte l'ensemble des propriétés exprimées sur le méta-modèle. Le langage défini (niveau M2) permet ensuite d'exprimer et de produire des modèles.

4.2.1.4. Méta-méta-modèle

Un méta-méta-modèle est un modèle qui décrit un langage de méta-modélisation. En tant que modèle, il doit aussi être spécifié à partir d'un langage de modélisation de niveau supérieur. Afin de limiter le nombre de niveaux d'abstraction (au niveau M3), l'OMG [MOF 2006] a défini le méta-méta-modèle standard MOF (Meta-Object Facility) qui a la propriété de méta-circularité i.e., la capacité de se décrire lui-même. Aujourd'hui, on peut trouver d'autres variétés de MOF comme EMOF (Essential MOF Model) [MOF 2006], Ecore de Eclipse [Steinberg et al. 2008], Kermeta de l'IRISA [Drey et al. 2009], etc. pour construire des méta-modèles.

[16] Parcimonie : « Utilisation du minimum de causes élémentaires pour expliquer un phénomène » selon https://fr.wikipedia.org/wiki/Parcimonie

[17] Substituabilité : « Capture des informations nécessaires et suffisantes pour permettre de répondre aux questions que l'on se pose sur un aspect du système qu'il représente, exactement de la même façon que le système lui-même aurait répondu » selon [Jézéquel et al. 2012].

4.2.1.5. Transformation de modèles

Les opérations sur les modèles sont appelées « transformations » dans IDM. La transformation est considérée comme primordiale et est utilisée entre autres pour la génération de code, le refactoring, la migration technologique en établissant des ponts (mappings et traductions) entre les langages source et cible, etc. [Bézivin 2003]. Les transformations sont spécifiées au niveau des méta-modèles et elles peuvent être faites de façon automatique ou manuelle sur les modèles.

Par ailleurs, Turki S. définit la transformation de modèles comme étant la génération d'un ou de plusieurs modèles cibles à partir d'un ou de plusieurs modèles sources conformément à une définition de règles de transformation [Turki 2008]. Définir des règles de transformation revient à décrire globalement comment un modèle spécifié à partir d'un méta-modèle source peut être transformé en un modèle spécifié à partir d'un méta-modèle cible.

Deux grands groupes de transformation de modèles [Jézéquel et al. 2012] sont donc considérés:

- Le « M2M » (Model To Model) : Utilisé pour générer un ou plusieurs modèles cibles à partir d'un ou plusieurs modèles sources de manière à ce que la transformation puisse être faite de façon automatique ou manuelle.
- Le « M2T » (Model To Text) : Utilisé pour générer différents types de représentations textuelles (xml, langage de programmation, etc.) à partir d'un ou plusieurs modèles sources.

Il existe quatre méthodes de transformation de modèles [Jézéquel et al. 2012] dans IDM, à savoir :

- La transformation « endogène » : le méta-modèle d'entrée et de sortie est le même, par exemple, pour générer du code en Java à partir d'un autre code en Java, on utilise le même méta-modèle qui est celui de Java ;
- La transformation « exogène » : le méta-modèle d'entrée est différent de celui de la sortie, par exemple, pour générer du code en Java à partir d'un diagramme UML (i.e. du M2T), on doit utiliser deux méta-modèles différents, un méta-modèle de UML comme source et un méta-modèle de Java comme cible;
- La transformation « verticale » : Dans le cas où on a besoin de changer de niveau d'abstraction, comme par exemple ce qui est fait dans l'approche MDA (Model Driven Architecture) [Soley 2000] définie par l'OMG en 2000, la transformation verticale permet

de passer du modèle d'exigence (CIM[18]) au modèle d'analyse et de conception (PIM[19]) ou du modèle d'analyse et de conception au modèle de code (PSM[20]) et inversement ;

- La transformation « horizontale » : La transformation se fait au même niveau.

Ces méthodes de transformation sont résumées par la Figure 34.

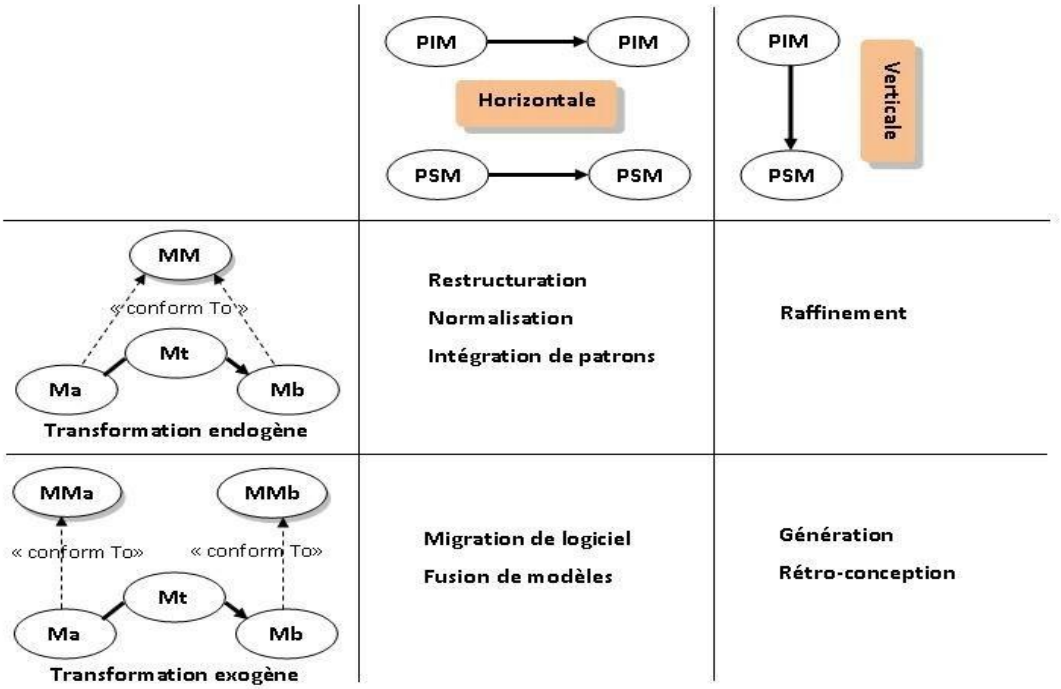


Figure 34.Méthodes de transformation et leurs principales utilisations [Combemale B. 2008], [Jézéquel et al. 2012]

4.2.2. Méta-modélisation et langage dédié (DSL)

Rappelons que l'OMG a défini le méta-modèle comme étant un modèle qui définit le langage d'expression d'un modèle [OMG 2004].

Un langage, que ce soit en linguistique (langage naturel) ou en informatique (langage de programmation ou de modélisation), est caractérisé par sa syntaxe et sa sémantique [Combemale B. 2008]. Il est donc défini selon le couple L:{Syntaxe, Sémantique}.

Un langage dédié (ou DSL) est représenté par la Figure 35, et est défini selon le t-uple {AS, CS*, M_{ac}*, SD, M_{as}*, M_{cs}} où il faut spécifier la syntaxe abstraite (AS), les syntaxes concrètes (CS), les mappings de la syntaxe abstraite vers la syntaxe concrète (M_{ac}), le domaine sémantique (SD) et

[18]CIM: Computation Independent Model (Modèle d'exigence)
[19]PIM: Platform Independent Model (Modèle d'analyse et de conception)
[20]PSM: Platform Specific Model (Modèle de code)

l'ensemble des mappings de la syntaxe abstraite vers le domaine sémantique (M_{as}) [Jézéquel et al. 2012].

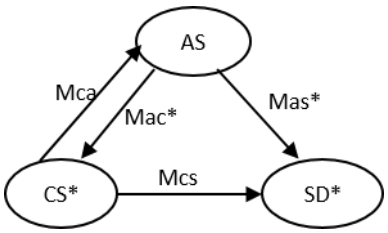


Figure 35. Composantes d'un langage [Jézéquel et al. 2012]

Ainsi, la méta-modélisation consiste à spécifier et à mettre en œuvre les différentes composantes du DSL, i.e. la syntaxe abstraite (AS), la ou les syntaxe(s) concrète(s) (CS) et la sémantique (SD) avec les différentes transformations (ou mappings) qui les relient, dans le but de pouvoir produire des modèles opérationnels.

4.2.2.1. Spécification d'une syntaxe abstraite

Définir la syntaxe abstraite du langage est généralement la première et la plus importante étape de la méta-modélisation car elle sert de base pour définir la ou (les) syntaxe(s) concrète(s) et la sémantique. La syntaxe abstraite permet de spécifier l'ensemble des concepts et des relations nécessaires pour pouvoir modéliser un domaine de discours. Elle décrit d'une part, la structure commune des éventuels modèles possibles à l'aide d'un langage de méta-modélisation, d'autre part, les propriétés liées à la sémantique statique qui n'ont pas été prises en compte par le méta-modèle, à l'aide d'un langage de contrainte.

Des langages de méta-modélisation intégrés dans des IDE (Environnement de Développement Intégré) sont actuellement disponibles sur le marché afin de pouvoir spécifier des syntaxes abstraites, comme par exemple, Eclipse-EMF avec son méta-méta-modèle Ecore [Steinberg et al.2008], GME avec son méta-méta-modèle MetaGME [Molnar et al. 2007], AMMA avec son méta-méta-modèle KM3 [Jouault et Bézvin 2006], XMF-Mosaic avec son méta-méta-modèle Xcore [Combemale et al. 2006], Kermeta [Drey et al. 2009], Software factories [Greenfield et al. 2003], et les langages issus du langage de contrainte OCL [Warmer et Kleppe1998]. Les langages de méta-modélisation sont en général, des variantes du standard MOF [MOF 2006] (Figure 36).

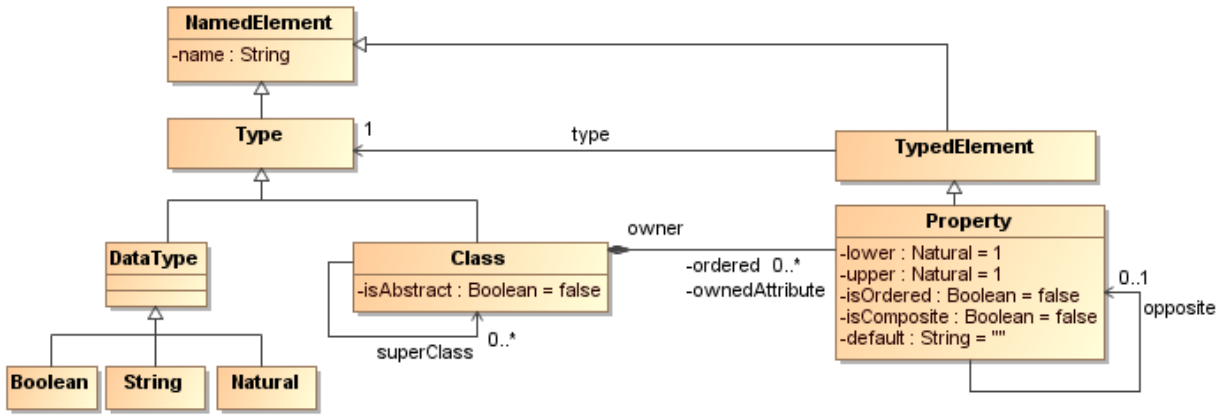


Figure 36. Concepts principaux de méta-modélisation (EMOF 2.0)

Dans le cadre du présent travail de recherche, nous allons utiliser le méta-méta-modèle Ecore représenté sur la Figure 37 avec les outils associés fournis par le framework EMF (Eclipse Modeling Framework) pour construire les modèles permettant de construire l'initialisation et l'observation des modèles de simulation.

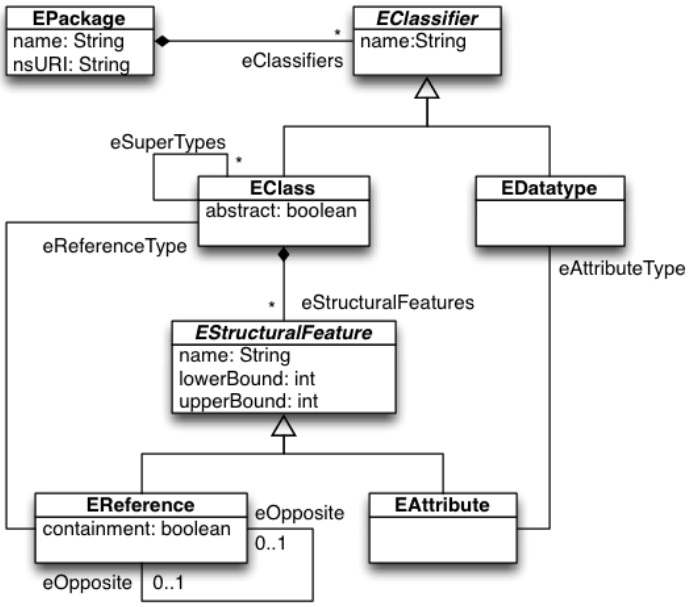


Figure 37. Représentation UML du méta-méta-modèle Ecore [MOF 2006]

4.2.2.2. Spécification d'une syntaxe concrète

Définir une syntaxe concrète revient à définir un mapping M_{ac} de la Figure 35 du DSL avec la spécification d'une grammaire que l'utilisateur va utiliser directement pour construire son modèle.

La syntaxe concrète peut être soit textuelle (sous forme de pseudo-code), soit graphique (comme les diagrammes UML par exemple) selon les besoins des utilisateurs. Elle permet à l'utilisateur (thématicien, modélisateur ou informaticien) d'exprimer de manière formelle un modèle puis d'instancier et de manipuler les concepts et les relations définis dans la syntaxe abstraite. Le modèle ainsi obtenu est conforme au méta-modèle du langage de modélisation. Actuellement, des projets permettant de définir des syntaxes concrètes sont mis en œuvre et bien maîtrisés. Ils ont permis de normaliser la construction des syntaxes concrètes. Entre autres, des outils basés sur le framework EMF [Steinberg et al. 2008] comme XText[21] et EMFText[22] permettent de générer des éditeurs textuels et GEF[23], GMF[24], Sirius[25], TOPCASED[26] ou OBEO Designer[27] permettent de générer des éditeurs graphiques.

Comme nous utilisons le méta-méta-modèle Ecore qui est issu du projet Eclipse Modeling Project[28] (EMP) pour la spécification des syntaxes abstraites de nos langages dédiés, nous allons utiliser les outils XText et GMF pour développer nos DSL d'initialisation et d'observation et pour montrer la généricité de notre approche.

- **Présentation de XText**

XText est un framework permettant de développer facilement des DSL et de générer un éditeur textuel pour les utilisateurs, à partir d'un méta-modèle Ecore comme présenté par la Figure 38 et d'un fichier contenant la description de la grammaire du DSL comme montré par la Figure 39. Cette grammaire est comparable à celle de type EBNF[29] (ou Extended Backus-Naur Form). XText est intégré dans l'environnement de développement Eclipse. Il permet de fournir un environnement convivial aux modélisateurs avec une coloration syntaxique personnalisable, l'auto-complétion sur les éléments du DSL spécifié, la détection d'erreurs automatique, etc. (Figure 40). Il couvre également tous les aspects d'un IDE moderne comme le parseur, le compilateur ou l'interpréteur.

[21]https://eclipse.org/Xtext/
[22] http://www.emftext.org/
[23]https://eclipse.org/gef/
[24]http ://www.eclipse.org/modeling/gmf/
[25]http ://www.eclipse.org/sirius/
[26]http://www.topcased.org/
[27]http://obeo.fr/pages/obeo-designer/fr/
[28]http://www.eclipse.org/modeling/
[29]https://www.cs.cmu.edu/~pattis/misc/ebnf.pdf

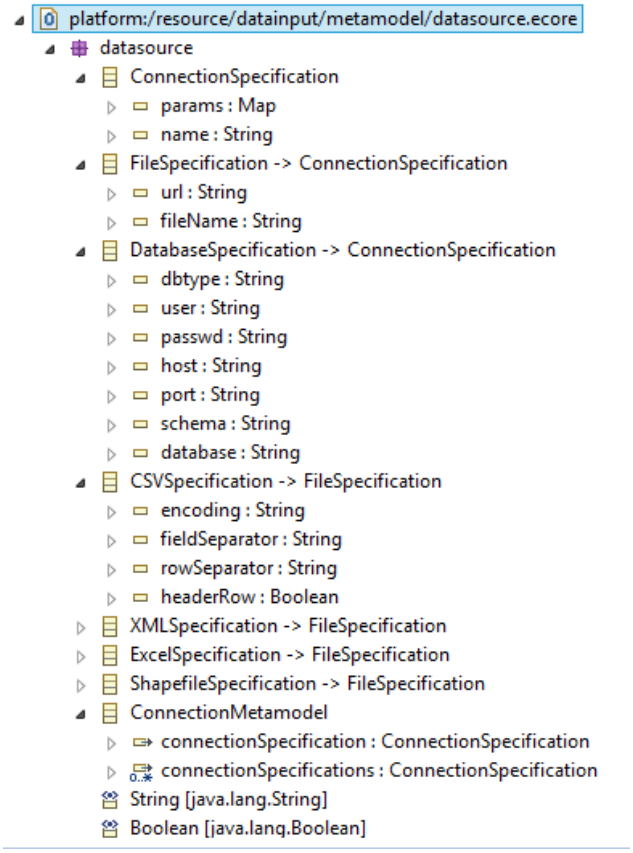


Figure 38. Exemple de méta-modèle Ecore

```
grammar org.xtext.datainput.datasource.Datasource with org.eclipse.xtext.common.Terminals

import "platform:/resource/datainput/metamodel/datasource.ecore"
import "http://www.eclipse.org/emf/2002/Ecore" as ecore
import "http://www.eclipse.org/xtext/common/JavaVMTypes" as jvmTypes


ConnectionMetamodel returns ConnectionMetamodel:
    {ConnectionMetamodel}
    'DatasourceSpecification'
    '{'
        connectionSpecifications+=ConnectionSpecification*
    '}';

ConnectionSpecification returns ConnectionSpecification:
    DatabaseSpecification | CSVSpecification | XMLSpecification | ExcelSpecification | ShapefileSpecification;

DatabaseSpecification returns DatabaseSpecification:
    {DatabaseSpecification}
    'DatabaseSpecification' name=STRING
    '{'
        ('dbtype' ':' dbtype=STRING)?
        ('user' ':' user=STRING)?
        ('passwd' ':' passwd=STRING)?
        ('host' ':' host=STRING)?
        ('port' ':' port=STRING)?
        ('schema' ':' schema=STRING)?
        ('database' ':' database=STRING)?
    '}';
```

Figure 39. Exemple de définition d'une grammaire utilisant le méta-modèle Ecore avec XText

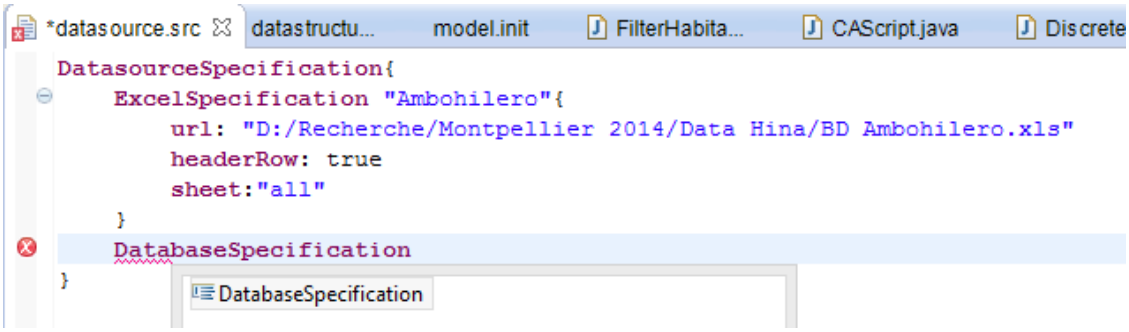


Figure 40. Exemple d'éditeur textuel généré à partir de XText

- **Présentation de GMF**

GMF (Graphical Modelling Framework) est un framework Eclipse permettant de créer un éditeur graphique spécifique. Son principe d'utilisation est conforme à l'architecture MVC (Model - View - Controller). Il utilise six fichiers pour créer un éditeur graphique. Ces fichiers doivent être spécifiés par étape en suivant le processus représenté par le tableau de bord présenté par la Figure 41.

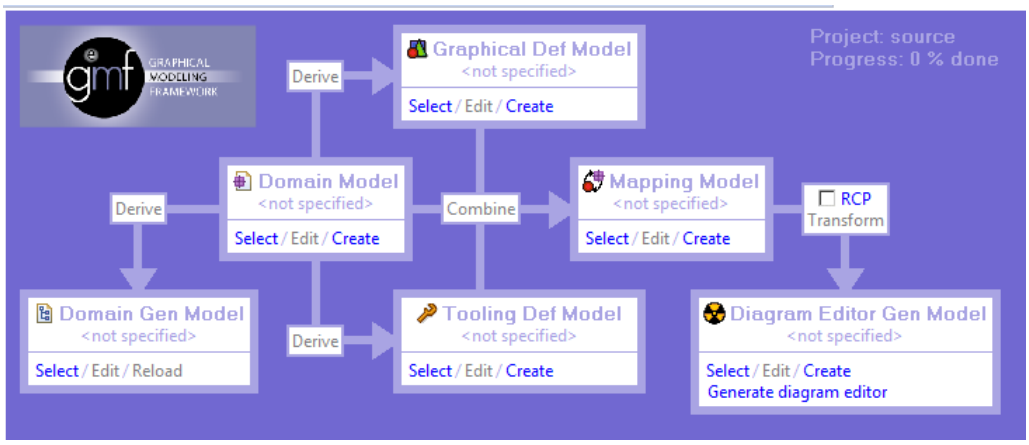


Figure 41. Tableau de bord pour le développement d'un éditeur graphique avec GMF

En effet, six étapes sont nécessaires pour générer un éditeur graphique comme ce qui est présenté par exemple par la Figure 47, à savoir :

1. Domain Model : cette étape permet de spécifier le métamodèle que l'utilisateur voudrait utiliser pour créer l'éditeur graphique (par exemple, Figure 38);
2. DomainGenModel (.genmodel) : cette étape permet de générer le code du modèle de domaine (Figure 42) ;

```
<?xml version="1.0" encoding="UTF-8"?>
<genmodel:GenModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:ecore="http://www.eclipse.org/emf/20
    xmlns:genmodel="http://www.eclipse.org/emf/2002/GenModel" modelDirectory="/datainput/src" modelPluginID="datai
    rootExtendsClass="org.eclipse.emf.ecore.impl.MinimalEObjectImpl$Container" importerID="org.eclipse.emf.importe
    containmentProxies="true" complianceLevel="6.0" operationReflection="true" importOrganizing="true">
  <foreignModel>datasource.ecore</foreignModel>
  <genPackages prefix="Datasource" disposableProviderFactory="true" ecorePackage="datasource.ecore#/">
    <genDataTypes ecoreDataType="datasource.ecore#//String"/>
    <genDataTypes ecoreDataType="datasource.ecore#//Boolean"/>
    <genDataTypes ecoreDataType="datasource.ecore#//Map"/>
    <genClasses ecoreClass="datasource.ecore#//ConnectionSpecification">
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute datasource.ecore#//ConnectionSpecification/p
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute datasource.ecore#//ConnectionSpecification/r
    </genClasses>
    <genClasses ecoreClass="datasource.ecore#//FileSpecification">
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute datasource.ecore#//FileSpecification/url"/>
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute datasource.ecore#//FileSpecification/fileNam
    </genClasses>
    <genClasses ecoreClass="datasource.ecore#//DatabaseSpecification">
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute datasource.ecore#//DatabaseSpecification/dbt
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute datasource.ecore#//DatabaseSpecification/use
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute datasource.ecore#//DatabaseSpecification/pas
```

Figure 42. Exemple de fichier généré du modèle de domaine

3. GraphicalDefModel (.gmfgraph) : cette étape permet de définir les éléments graphiques du modèle de domaine (Figure 43) ;

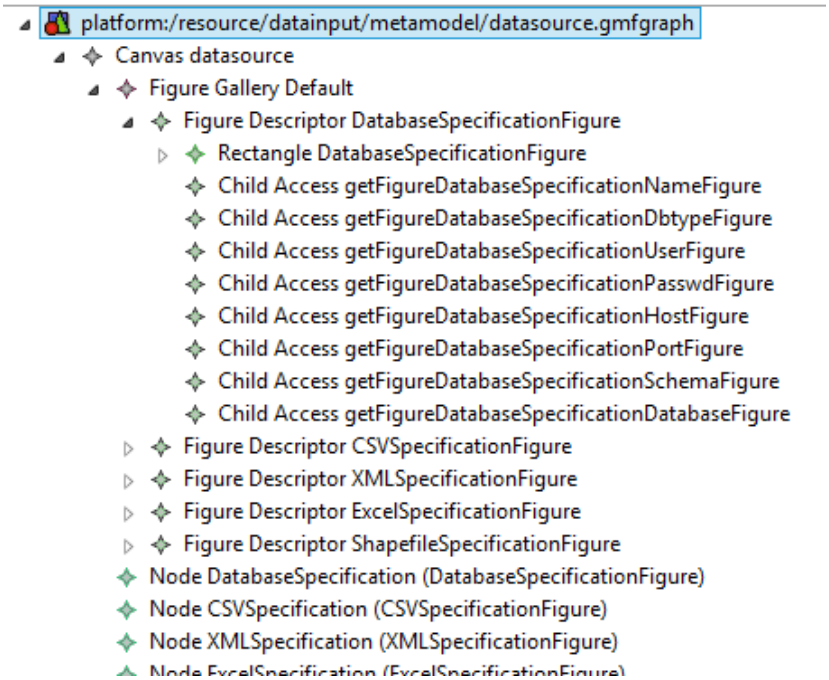


Figure 43. Exemple de fichier pour la définition d'éléments graphiques

4. ToolingDefModel (.gmftool**)** : cette étape permet de définir la palette d'outils que l'utilisateur pourra utiliser dans l'éditeur graphique (Figure 44) ;

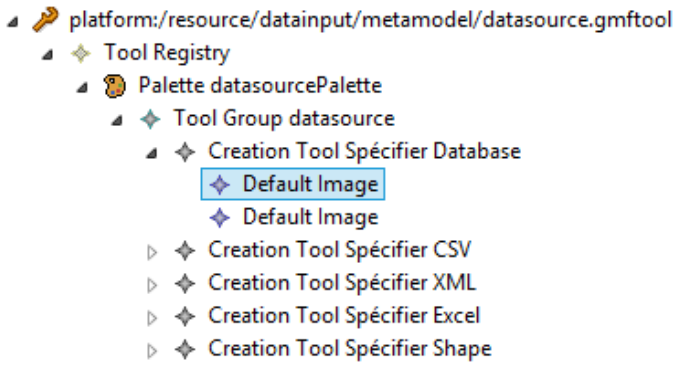


Figure 44. Exemple de fichier pour la définition de palette d'outils

5. MappingModel (.gmfmap) : cette étape permet de lier le modèle de domaine, le modèle graphique (.gmfgraph) et le modèle des boîtes à outils (Figure 45) ;

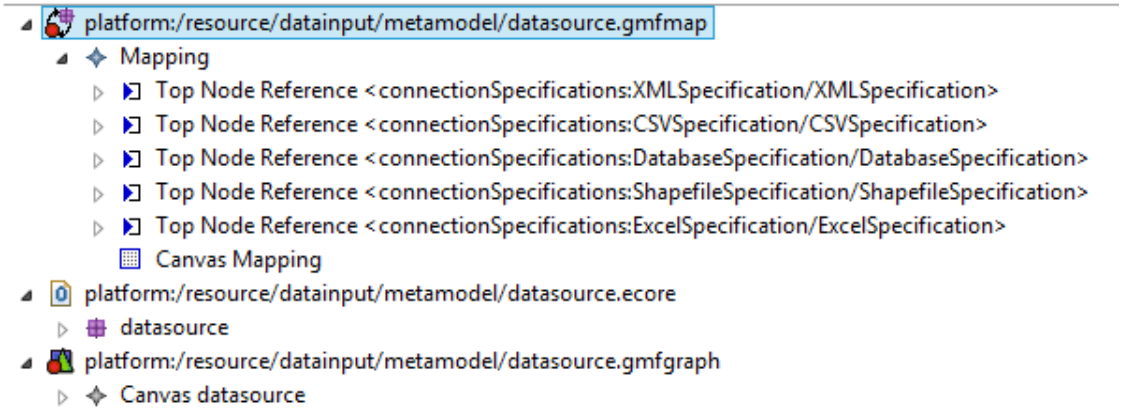


Figure 45. Exemple de fichier généré pour la liaison des différents modèles de domaine, graphique et de palette

6. Diagram EditorGenModel (.gmfgen) : cette étape finale permet de générer l'éditeur graphique GMF en plus du code généré parle fichier.genmodel à l'étape 2 (Figure 46).

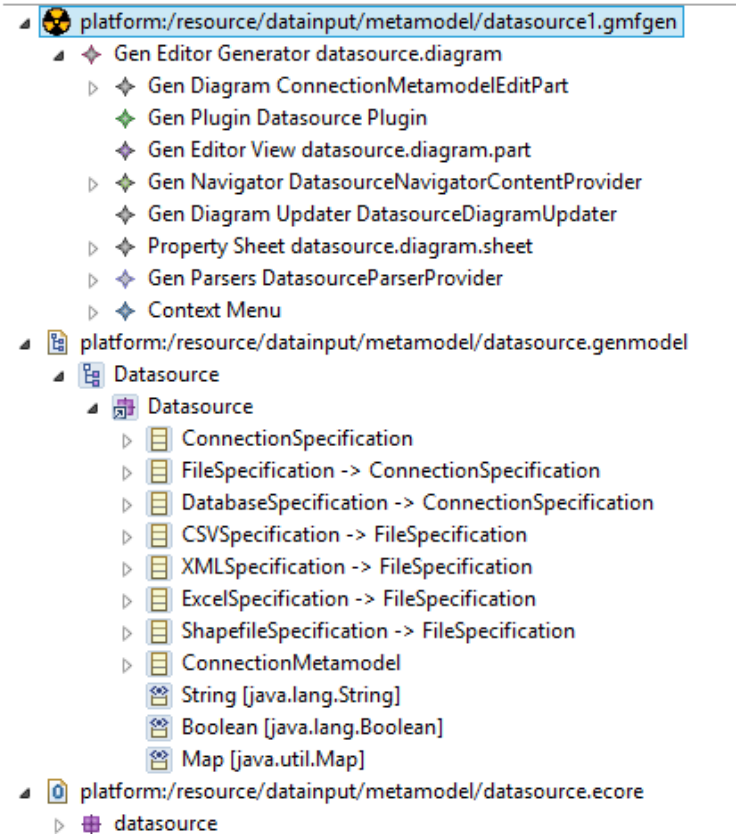


Figure 46. Exemple de fichier pour la génération d'un éditeur graphique GMF

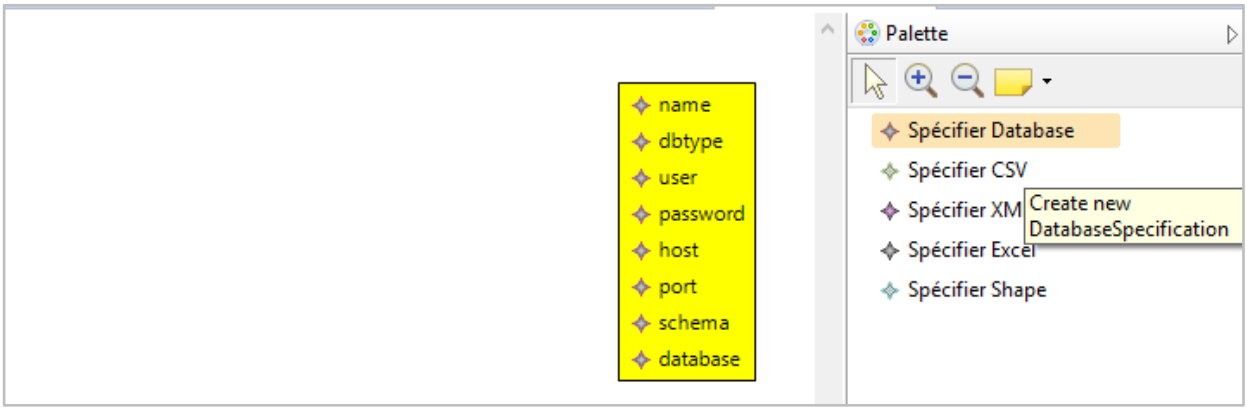


Figure 47. Exemple d'éditeur graphique généré par GMF

4.2.2.3. Spécification d'une sémantique

Pour que les modèles créés à partir du DSL soient opérationnels et non seulement descriptifs, il est aussi nécessaire de formaliser sa sémantique, i.e., le mapping M_{as}de la Figure 35, entre la syntaxe abstraite (AS) et la sémantique.

En effet, la sémantique d'un DSL permet de définir de manière unique et sans ambiguïté le sens des éléments des modèles construits à partir du langage [Jézéquel et al. 2012]. On distingue deux types de sémantique, à savoir :

- La sémantique statique : exprimée en général sur la syntaxe abstraite à l'aide d'un langage de contrainte comme OCL, afin de définir les règles et les contraintes que doivent respecter les concepts et leurs relations lors d'une instanciation, par exemple, la multiplicité, les invariants, les pré- ou post-conditions. Les règles sont ensuite vérifiées lors de la compilation du programme.
- La sémantique comportementale : exprimée sous forme de modèle, elle permet de décrire le comportement du programme. Winkel décrit trois façons de définir une sémantique comportementale, soit de manière opérationnelle, dénotationnelle ou axiomatique [Winskel 1993].Cette formalisation permet ensuite de vérifier le programme lors de son exécution.

Des outils comme XTend[30] , XPand[31], Acceleo[32], ATL[33], QVTO[34] ou Kermeta[35]dédiés à la transformation de modèles (M2M ou M2T) peuvent aussi bien être utilisés pour spécifier les sémantiques de nos DSL afin de générer du code ou des modèles que nous souhaitons. Et les outils que nous avons énumérés précédemment, notamment XText et GMF sont capables d'interpréter et de traiter automatiquement les sémantiques statique et comportementale définies lors de la méta-modélisation à l'aide de ces outils de transformation de modèles.

Pour la sémantique comportementale de nos DSL, nous avons utilisé le langage XTend [Bettini 2013] afin de générer du code à partir de XText et nous avons également utilisé la technique proposée par Jan Köhnlein[36] pour synchroniser et combiner les syntaxes concrètes graphique fournie par GMF et textuelle fournie par XText.

[30] https://eclipse.org/xtend/
[31] https://eclipse.org/modeling/m2t/?project=xpand
[32]https://eclipse.org/acceleo/
[33] https://eclipse.org/atl/
[34] https://wiki.eclipse.org/QVTo
[35] http://www.kermeta.org/
[36] http://koehnlein.blogspot.com/2009/06/synchronized-editors-with-tmfxtext-and.html

- **Le langage XTend**

XTend est un langage de programmation proche de Java permettant d'écrire de manière simple un générateur de code et il est déjà intégré dans l'outil XText. En effet, la génération de code est possible grâce à l'implémentation de l'interface « IGenerator » (Figure 48) qui a pour charge de générer du code à partir de XTend. Lors de l'utilisation du DSL, chaque déclaration d'entité ou de relation du méta-modèle considéré par le langage génère automatiquement le code d'application souhaité par l'utilisateur comme par exemple du code en Java, en SmallTalk, en XML ou d'un fichier de configuration, d'un fichier texte, etc.

```
/*
 * Copyright (c) 2011 itemis AG (http://www.itemis.eu) and others.
package org.eclipse.xtext.generator;

import org.eclipse.emf.ecore.resource.Resource;


/**
 * @author Sven Efftinge - Initial contribution and API
 */
public interface IGenerator {

    /**
     * @param input - the input for which to generate resources
     * @param fsa - file system access to be used to generate files
     */
    public void doGenerate(Resource input, IFileSystemAccess fsa);

    /**
     * @since 2.4
     */
    public static class NullGenerator implements IGenerator {
        public void doGenerate(Resource input, IFileSystemAccess fsa) {}
    }
}
```

Figure 48. Interface permettant de générer du code dans XTend

4.2.3. Quelques applications d'IDM dans la littérature

Selon [Bézivin 2003], l'IDM est utilisée dans de nombreux domaines de l'informatique, entre autres, le web sémantique, l'intégration de données hétérogènes, les entrepôts de données, le E-commerce, l'évolution des applications, etc. Le principe fondamental est de définir des DSL, de les maîtriser et de spécifier des mises en correspondance entre les méta-modèles source et destination selon [El Hamlaoui 2012] afin de générer un modèle conforme au méta-modèle destination à partir d'un modèle source. Une des premières approches de l'IDM date de 1988 et consiste à réaliser les outils CASE (Computer Aided Software Engineering) d'après [Premkumar et al. 1995], lequel désigne aujourd'hui les environnements de développement comme Topcased,

ArgoUML, BOUML, EclipseUML, StarUML, permettant de générer des codes sources à partir des modèles UML. Selon [Soley etal. 2000], MDA est une variante particulière de l'IDM adopté par l'OMG dans le but de résoudre les problèmes actuels de construction et d'évolution de systèmes d'information auxquels font face les industries. MDA consiste à passer des modèles appelés PIM, indépendants des plateformes techniques, aux modèles appelés PSM, dépendants des plateformes techniques pour faciliter la génération de code vers une plateforme technique choisie. Plusieurs DSL ont déjà été proposés pour les systèmes complexes. Entre autres, Hill a proposé un DSL permettant de configurer des systèmes en général et particulièrement des systèmes modulaires [Hill 2006]. Le réseau CLASYCO[37] cite quelques DSL dans son site web, à savoir : Ocelet (Cirad), qui utilise des graphes d'interaction pour la description des dynamiques spatiales et appliqué pour décrire la dynamique de la mangrove amazonnienne, MGS (U-PEC / CNRS) pour la description de systèmes dynamiques à structures dynamiques en biologie, SIMPOP pour la modélisation multi-niveaux de dynamiques urbaines, Scalation pour la modélisation discrète de phénomènes.

La technique d'IDM est connue depuis les années 90 à cause de l'accroissement exponentiel de la complexité des systèmes informatiques, mais son utilisation dans la modélisation des domaines thématiques comme la biologie, l'agriculture, l'élevage ou la socio-écologie n'en est qu'à ses débuts. Aujourd'hui, elle permet de faire rapprocher les informaticiens et les experts de domaine dans la réalisation de modèles spécifiques, et des conférences autour de la modélisation scientifique et la génération de code informatique sont actuellement de plus en plus organisées. Dans ce sens, Bruel et al. ont élaboré des expériences concrètes qui consistent à modéliser de façon participative, un système de culture, afin d'évaluer l'apport de l'IDM sur les problèmes scientifiques. Par la suite, ils ont affirmé que l'intégration des concepts, spécifiques à un domaine et compréhensibles par les thématiciens, dans le processus complet de modélisation[38] peut améliorer considérablement la qualité du modèle développé et la productivité des experts du domaine. Cette intégration peut également aider à combler l'écart existant entre les problèmes scientifiques des thématiciens et son implémentation informatique [Bruel et al. 2015].

Ces faits nous poussent à exploiter cette nouvelle technologie de l'informatique pour mettre en œuvre le processus d'initialisation et d'observation de modèles de SES.

[37] http://www.spatial-computing.org/dsl/start

[38] Processus complet de modélisation: à partir de l'analyse, la conception du modèle conceptuel, le développement du logiciel, l'exécution de la simulation jusqu'à la visualisation

4.3. Mise en correspondance entre les concepts d'IDM et le processus d'initialisation et d'observation de modèles de simulation

Nous proposons de reformuler les questions d'initialisation et d'observation des modèles de SES en un problème de spécification de transformations entre des données et des structures de données. En effet, ce processus implique des transformations entre les données des thématiciens et les structures de données d'entrée du modèle, les structures de données d'entrée du modèle et celles utilisées pour la simulation, les structures de données du modèle et celles de sortie, et enfin, les structures de données de sortie et les indicateurs. Par rapport à la définition de Minsky représentée par la Figure 49, d'une part les systèmes "B" à modéliser correspondant au niveau M_0 de la Figure 33 sont respectivement les sources de données, les structures de données d'entrée, les structures de données du modèle de simulation, les observables, les indicateurs et les transformations entre eux ; d'autre part, les modèles "A" correspondant au niveau M_1 de la Figure 33 sont respectivement les descriptions des sources de données et de leurs contenus, les descriptions des structures de données d'entrée indépendamment des supports, celles des structures de données du modèle à partir desquelles les structures de données de sortie sont extraites, celles des structures des indicateurs, ainsi que les spécifications des transformations.

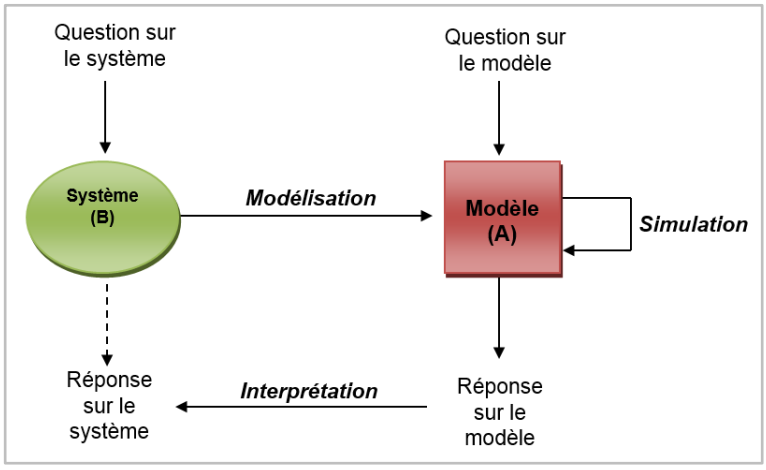


Figure 49. Système et modèle selon Minsky

Pour être en mesure de créer les différents modèles précités correspondant au niveau M_1 de la Figure 33, nous devons pour chaque système (niveau M_0 de la Figure 33), fournir respectivement un DSL (niveau M_2 de la Figure 33).

Chaque DSL est constitué d'abord, d'une syntaxe abstraite sous la forme d'un méta-modèle pour exprimer l'ensemble des concepts et les relations utilisés dans le système ; ensuite d'une ou de plusieurs syntaxes concrètes qui sont mises à la disposition des utilisateurs pour manipuler les

concepts de la syntaxe abstraite afin de construire des modèles et enfin, de la sémantique pour donner un sens précis aux modèles construits afin de générer éventuellement du code informatique.

Ces méta-modèles doivent être spécifiés avec un méta-méta-modèle (niveau M_3 de la Figure 33). Ayant choisi EMF comme outil de développement, nous allons utiliser le méta-méta-modèle Ecore [Steinberget al. 2008] qui est une variante existante du MOF.

Ainsi, la mise en correspondance complète entre notre proposition et la pyramide de modélisation de l'OMG est illustrée par la Figure 50.

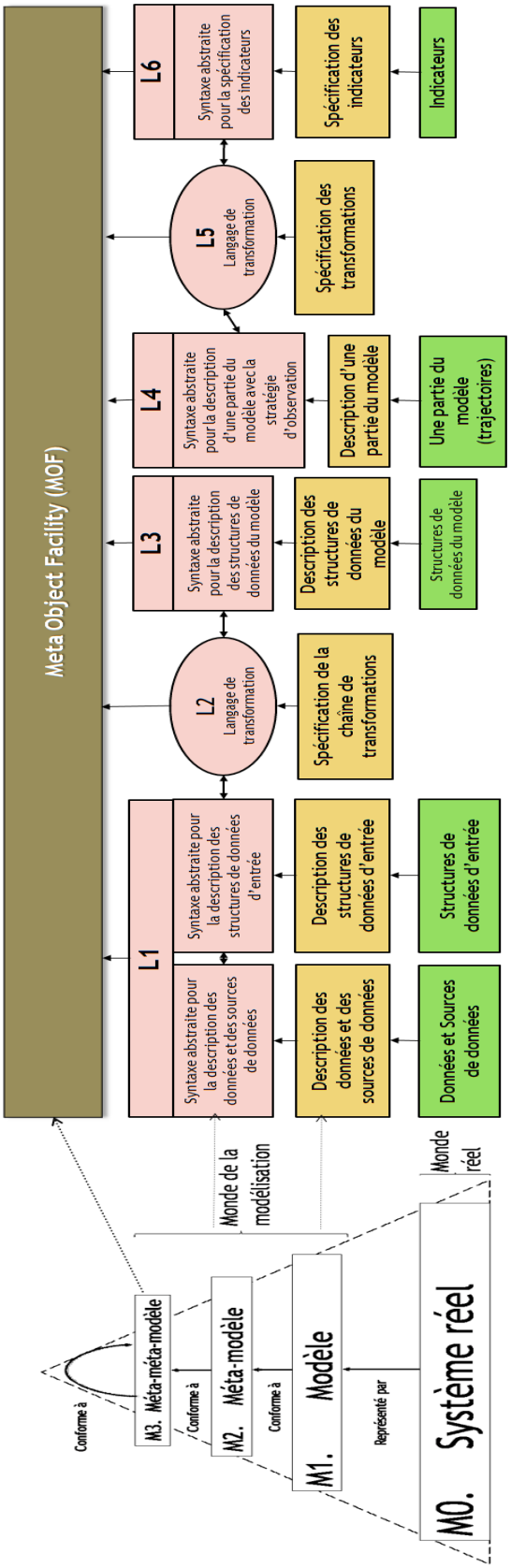


Figure 50. Mise en correspondance de la pyramide de modélisation et le processus d'initialisation et d'observation

Nous avons ainsi besoin de spécifier les syntaxes abstraites correspondantes aux DSL d'initialisation et d'observation de modèle avec le méta-méta-modèle Ecore :

- Le langage L1: utilisé pour décrire les différentes sources de données et leurs structures. Par défaut, le langage L1 ne prend pas en compte la notion de temps. Les données de L1 sont utilisées pour construire l'état initial du modèle de simulation, i.e. la structure de données du modèle à sa date initiale, en tant que paramètres ou séries chronologiques aussi explicites.
- Le langage L3 : utilisé pour décrire le modèle de simulation. Le modèle de simulation est considéré comme une fonction continue du temps dans les structures de données représentant l'état du modèle à une date donnée. Son état initial est donc considéré comme une description partielle de la fonction du temps. Par conséquent, une simulation est considérée comme un procédé de réalisation de cette fonction du temps sur un intervalle donné.
- Le langage L2 : utilisé pour spécifier la chaîne de transformations possibles entre les données décrites par L1 et le modèle de simulation décrit par L3.
- Le langage L4 : utilisé pour décrire des sous-parties du modèle que l'on veut observer et suivre pendant la simulation avec une stratégie d'observation.
- Langage L6 : utilisé pour spécifier la structure des indicateurs ainsi que leurs présentations du côté des utilisateurs.
- Le langage L5: utilisé pour spécifier la chaîne de transformations possibles à partir des observables du modèle pour produire les indicateurs souhaités.

Les prochaines étapes consistent ensuite à définir pour chaque DSL, une ou des syntaxes concrètes (CS) pour les utilisateurs puis le domaine sémantique (SD) pour son interprétation. Le résultat attendu est la génération automatique de code pour obtenir effectivement les différentes structures de données et les indicateurs souhaités, avec les diverses transformations.

4.4. Quelques éléments exploitables pour les processus d'initialisation et d'observation de modèles de simulation.

La mise en œuvre d'un méta-modèle avec les transformations éventuelles nécessite la compréhension et la maîtrise du domaine ou du système que l'on veut modéliser. Afin de proposer un outil générique pour le processus d'initialisation et d'observation de modèles de SES en particulier, il est nécessaire de pouvoir identifier les concepts nécessaires pour pouvoir spécifier :

- des données issues des différentes sources hétérogènes fournies par les thématiciens et leurs structures ;
- des différents éléments apparaissant dans les modèles tels que les éléments biophysiques et sociaux comme l'espace, le temps, les foyers, les ressources, les activités, les territoires, les institutions, etc. utilisés dans le modèle de simulation,
- des sorties souhaités avec leurs présentations visuelles lors de la simulation.

Des travaux ont été déjà menés par rapport à ces besoins. Entre autres, pour la représentation et l'exploration des données et de leurs sources, la notion de schéma de données issue de différentes sources de données comme les bases de données, les fichiers XML, CSV ou Excel, etc. est très utilisée dans la littérature. Ces schémas permettent de représenter les structures de données stockées dans les sources de données et peuvent être accessibles et exploitables pour générer les données via des pilotes de connexion. Ces différentes structures de données peuvent ensuite être transformées en d'autres formalismes pour d'autres besoins éventuels, comme il est expliqué dans [Halpin et al. 1995], [Golshani et al. 1998] en utilisant les types de transformations proposés par l'IDM (le M2M et le M2T).

Pour la description d'un modèle de SES, plusieurs travaux basés sur les ontologies [Müller et al. 2011] visent à représenter de manière générique, les différents éléments ou concepts employés dans le domaine de discours du modèle de simulation. Par exemple, nous avons proposé une extension de la logique descriptive [Müller et al. 2011], afin de représenter les territoires et les institutions de manière générique dans le contexte des SES. Et puisqu'on parle de simulation, (i.e., l'évolution des modèles de simulation dans le temps) des recherches ont été aussi faites, dans ce sens, pour représenter les données temporelles. Dans la littérature, nous pouvons constater que deux types d'ontologie sont en compétition pour représenter ces dernières. D'une part, l'ontologie temporelle [Gayral et al. 1992], [Dermott 1982], fondée sur les entités temporelles et qui a été par exemple appliquée dans le monde du web à travers le [Pan et Hobbs 2005]. D'autre part, l'ontologie évènementielle [Dermott 1982], fondée sur les entités évènementielles. Plusieurs variantes basées sur les logiques temporelles sont aussi utilisées pour représenter les données temporelles, par exemple [Allen 1981] a essayé d'associer un intervalle à tout évènement et de traduire toute information temporelle entre deux évènements par des relations temporelles. Dermott a distingué à partir de la théorie d'Allen [Allen 1981], une ontologie strictement temporelle et une ontologie évènementielle [Dermott 1982]. Kayser a exploité la logique des prédicats en ajoutant les entités temporelles [Kayser 1990] et Davidson les entités évènementielles

[Davidson 1981]. Kamp a basé sa représentation du temps sur les concepts d'évènement et d'état [Kamp 1981].

Pour l'analyse et la visualisation des modèles de simulation, certains travaux comme ceux de Gary P. visent à proposer une métadonnée standardisée afin de pouvoir enregistrer et explorer efficacement les sorties d'un modèle de simulation [Gary et al. 2014].

4.5. Application : Le modèle ECEC comme « fil rouge »

Bien que l'approche que nous avons proposée, a été appliquée pour initialiser et observer le modèle Mirana [Aubert et al. 2010], par la suite, afin de mieux illustrer les processus d'initialisation et d'observation des modèles de simulation et afin de montrer la généricité de l'approche, nous allons considérer un modèle écologique très simple dénommé ECEC [Pepper et Smuts 2002] comme « fil rouge ».

4.5.1. Présentation du modèle ECEC

ECEC (ou Evolution of Cooperation in an Ecological Context) [Pepper etSmuts 2002] est un modèle multi-agent très simple qui permet de simuler l'évolution de la coopération des acteurs dans le contexte écologique. Il est utilisé par beaucoup de chercheurs [Bommel 2011] pour expliquer de manière simple le fonctionnement des modèles à base d'agents (ou ABM pour Agent-based modeling en anglais). Son objectif est de pouvoir étudier la survie des populations d'agents qui interagissent avec l'environnement et qui sont en compétition pour des ressources spatialisées. Deux types d'agents sont considérés dans le modèle ECEC, à savoir : les plantes et les prédateurs (Figure 51).

- **Les plantes (la végétation)**

  Chaque plante est créée une seule fois à un emplacement fixe dans l'environnement au début de la simulation. Elle ne bouge pas mais elle a un comportement de croissance et elle sert de nourriture pour les prédateurs.

  La croissance des plantes fait varier la biomasse qui représente la quantité de nourriture disponible pour les prédateurs. A chaque pas de temps, la biomasse augmente en fonction de la courbe de croissance logistique $X_{t+1} = X_t + rX_t\left(1 - \frac{X_t}{K}\right)$ telle que :

  X_t : $quantité\ de\ biomasse\ au\ temps\ t$

  K: $capacité\ de\ charge\ du\ milieu$

r: *taux de reproduction*

- **Les prédateurs**

A chaque pas de temps et indépendamment de leur mobilité, les prédateurs perdent une certaine quantité d'énergie en fonction de leur taux de métabolisme. Dans le modèle, ce taux est le même pour tous les prédateurs.

Si la quantité d'énergie atteint zéro, le prédateur meurt. Cependant, s'il atteint un seuil de fécondité, il se reproduit et il y a création d'une progéniture avec les mêmes caractères héréditaires que son origine. La quantité d'énergie du prédateur parent est ensuite réduite à son énergie initiale et sa nouvelle progéniture va occuper l'endroit libre le plus proche.

Pour se nourrir, chaque prédateur examine son emplacement et les cellules voisines. Il choisit et se déplace à l'endroit où la plante contient la plus grande quantité de biomasse et pas encore occupé par un autre prédateur. Il augmente ensuite son niveau d'énergie tout en réduisant la même quantité de biomasse à la plante concernée. Si aucun endroit ne lui correspond, il se déplace aléatoirement, à un endroit adjacent libre.

Les prédateurs sont catégorisés dans deux types de stratégies de consommation, à savoir:

- La stratégie dite « restrained », où les prédateurs ne récoltent à chaque nourriture que 50% de l'énergie produite par la plante ;
- La stratégie dite « unrestrained », où ils mangent 99% de l'énergie fournie par la plante.

Ce taux de récolte est inférieur à 100% pour que les plantes ne soient pas détruites définitivement mais puissent continuer à croître après le passage d'un prédateur.

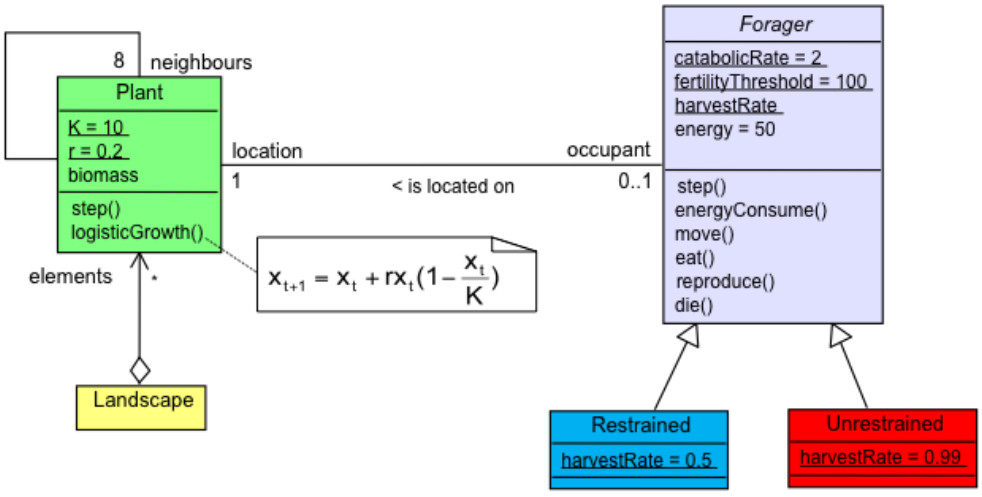


Figure 51. Diagramme de classe du modèle ECEC [Bommel et al. 2012]

4.5.2. Initialisation du modèle ECEC

Nous avons besoin d'initialiser et de paramétrer l'espace, les plantes ainsi que les prédateurs.

- Pour l'espace, il faut spécifier :
  - La taille de la grille ;
  - La topologie des voisins.
- Pour les plantes:
  - Le nombre de plantes (aléatoirement, selon un nombre minimum, selon le nombre de cellules, ou défini directement dans un support de données) ;
  - La taille maximale d'une plante (en unité de biomasse) ;
  - La quantité initiale de biomasse (X_0) ;
  - Le paramétrage du processus de croissance tel que :
    - Le taux de reproduction (r);
    - La capacité de charge du milieu (K).
- Pour les prédateurs :
  - Le nombre initial de prédateurs (de type « restrained » et de type « unrestrained ») ;
  - L'énergie initiale de chaque prédateur (en unité d'énergie) ;
  - Le taux de métabolisme ;
  - Le seuil de fécondité;
  - L'emplacement.

Les données dont on a besoin pour pouvoir initialiser le modèle ECEC peuvent provenir de différents projets et de différentes sources de données. Mais afin d'illustrer notre approche d'initialisation, nous allons considérer les sources de données suivantes, à savoir :

- Un fichier XML qui fournit les données pour construire l'espace ;

```
<Espace>
    <Nom>Grid</Nom>
    <Lignes>50</Lignes>
    <Colonnes>50</Colonnes>
    <Taille>10</Taille>
</Espace>
```

Figure 52. Données pour paramétrer l'environnement spatial du modèle ECEC

- Un fichier CSV (Figure 53) qui fournit le nombre initial de prédateurs de type « restrained » et de type « unrestrained »;

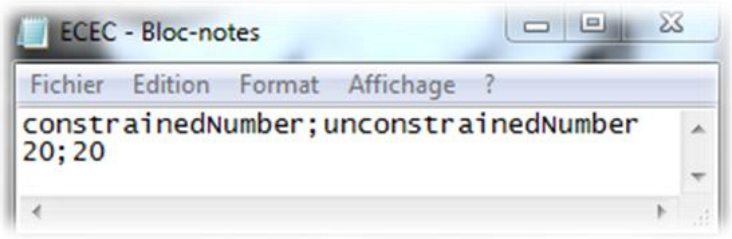


Figure 53. Données pour initialiser le nombre de prédateurs

- Un fichier Excel (Figure 54) qui fournit différents scénarios de paramètres de l'équation logistique utile pour la croissance des plantes;

scenario	K	r	X0
1	10	0,2	5
2	10	0,2	12
3	10	0,2	0,2

Plant class

Figure 54. Données pour paramétrer la fonction logistique

- Une base de données PostgreSQL (Figure 55) qui stocke les données utiles pour paramétrer les prédateurs;

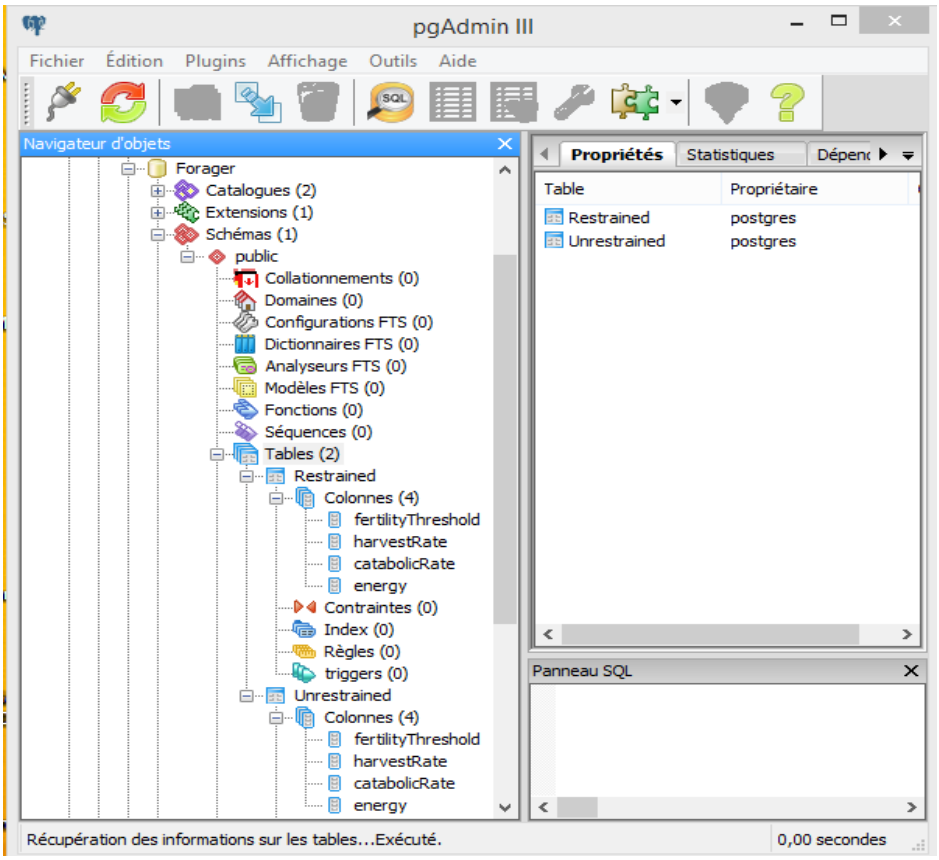


Figure 55. Données sur les caractéristiques des prédateurs

4.5.3. Observation du modèle ECEC

Pour illustrer le processus d'observation, nous allons partir des besoins qui peuvent se créer à partir du modèle ECEC. Supposons qu'à partir de la simulation du modèle ECEC, les thématiciens souhaitent traiter des questions sur la durabilité du système comme la résilience de l'environnement, la survie des populations d'agents ou la domination d'un type de population par rapport à l'autre (restrained et unrestrained), etc. Pour répondre à ces questions, ils peuvent être intéressés à suivre l'évolution de certains éléments du modèle, par exemple:

- L'évolution du nombre de prédateurs (« restrained » et/ou « unrestrained ») ;
- L'évolution de la quantité d'énergie totale des plantes ou des prédateurs ;
- L'évolution de la quantité d'énergie d'une ou d'un certain nombre de plantes ;
- L'évolution de la quantité d'énergie d'une ou de quelques prédateurs spécifiques ;

Pour pouvoir suivre ce qui se passe dans le modèle et afin de répondre à des questions sur son fonctionnement, il est nécessaire d'avoir les moyens, d'une part pour extraire, selon une stratégie d'observation, une partie des observables du modèle de simulation et de pouvoir la traiter afin de construire les indicateurs ; d'autre part de présenter ces indicateurs à l'écran et/ou de les sauvegarder sur des supports de données à des fins d'aide à la décision.

4.5.4. Implémentation du modèle ECEC dans la plateforme de M&S MIMOSA

Avant de procéder à l'initialisation et à l'observation du modèle ECEC, il est tout d'abord nécessaire de le construire. Nous avons implémenté le modèle ECEC avec la plateforme MIMOSA [Müller 2004]. Elle permet de construire des modèles de simulation en suivant le processus de modélisation formalisé par Farolfi [Farolfi et al. 2010]. Ce processus est divisé en quatre étapes de spécification (cf. 2.2.4). Dans cette section, nous allons seulement décrire le modèle conceptuel du modèle ECEC, ainsi que sa dynamique. La façon dont il sera initialisé et observé sera décrite dans les chapitres suivants avec la présentation des approches génériques d'initialisation et d'observation que nous proposons.

- **Spécification du modèle conceptuel du modèle ECEC**

Le modèle conceptuel du modèle ECEC peut être implémenté directement en utilisant du code Java dans la plateforme MIMOSA. Par ailleurs, MIMOSA offre également une interface graphique qui correspond à la T-box de la logique descriptive pour spécifier la structure statique

(ou le modèle conceptuel) du modèle ECEC à l'aide des catégories (ou concept en logique descriptive), des attributs et les relations (ou rôle, en logique descriptive).

- **Spécification de la dynamique du modèle ECEC**

Dans MIMOSA, la dynamique des éléments (ou entités) d'un modèle de simulation est basée sur le formalisme DEVS (ou simulation à évènements discrets) [Zeigler et al. 2000]. Nous n'allons pas approfondir ce formalisme, du fait que notre principal objectif ne concerne pas la dynamique d'un modèle de simulation mais plutôt l'observation de son effet sur le modèle de simulation. Néanmoins, le principe général de la dynamique est de décrire, avec des entités « DEVS atomique », l'ensemble des états possibles ainsi que les comportements (endogène[39] ou exogène[40]) que peuvent avoir chaque élément du modèle. C'est à la charge de l'ordonnanceur de la plateforme ensuite d'exécuter et de gérer la cohérence des dates d'exécution des différents comportements. Dans MIMOSA, différents formalismes peuvent être utilisés pour représenter la dynamique d'un modèle de simulation, à savoir :

  - Le diagramme d'état-transition ;
  - Les scripts tels que Python, Scheme, Smalltalk, Jess ;
  - Le langage Java (code en dur), car MIMOSA est implémentée en Java.

Pour le cas du modèle ECEC, nous avons implémenté les activités des agents avec des classes Java. Le code correspondant à cette implémentation est présenté en annexe C.

Par ailleurs, le modèle de simulation ECEC est basé sur un framework générique de MIMOSA dénommé « *Faritra* » [Müller 2015]. Il offre un méta-modèle générique des SES et permet de construire des modèles de simulation de différentes échelles, par exemple, au niveau d'un village, d'une commune, d'un district, ou d'une région. Le framework « *Faritra* » est basé sur les notions d'institution, d'organisation, de population, de territoire, de ressource et d'activité. Le modèle Mirana (cf. 2.2) a été aussi implémenté récemment sur la base de ce méta-modèle. Aussi, pour le fonctionnement du modèle ECEC, d'autres paramétrages supplémentaires sont donc nécessaires. Notamment, la spécification des institutions avec les différentes normes constitutives (Figure 56), la spécification des activités des agents (Figure 57), des contraintes sur les ressources (Figure 58), etc.

[39] Exécution du comportement sans influence (ou intervention) extérieure
[40] Exécution du comportement suite à des évènements extérieurs

Institution	Concept	Categorie
InstitutionEcology	biomass	ressource
InstitutionEcology	ecologist	acteur
InstitutionEcology	grow	action
InstitutionEcology	manageSpace	action
InstitutionEcology	habitat	lieu
InstitutionForager	forager	acteur
InstitutionForager	constrainedForager	acteur
InstitutionForager	unconstrainedForager	acteur
InstitutionForager	move	action
InstitutionForager	eat	action
InstitutionForager	live	action
InstitutionForager	energy	ressource
InstitutionForager	biomass	ressource
InstitutionForager	field	lieu
InstitutionForager	foragers	lieu

Figure 56. Données pour construire les institutions du modèle ECEC (cf. 2.2.4.1.a)

Agent	Activite	Implementation
Ecology	grow	ecec.activities.Growing
Ecology	manageSpace	faritra.activities.SpaceManagement
Forager	move	ecec.activities.Moving
Forager	eat	ecec.activities.Eating
Forager	live	ecec.activities.Living

Figure 57. Données pour paramétrer les activités des agents du modèle ECEC (cf. 2.2.4.1.c)

Agent	Ressource	Territoire	Lieu	Min	Max	Unite	Type
Ecology	biomass	habitat	habitat	1	100	unite	nombre
Forager	energy	foragers	constrainedForager	1	50	unite	nombre
Forager	energy	foragers	unconstrainedForage	1	50	unite	nombre

Figure 58. Données sur les ressources des agents du modèle ECEC (cf. 2.2.4.1.d)

Ainsi, le modèle ECEC est construit, il ne reste plus qu'à définir comment l'initialiser avec les données issues de diverses sources hétérogènes, puis la façon d'explorer la sortie de la simulation pour obtenir les indicateurs que l'on souhaite suivre.

4.6. Conclusion

Nous avons montré dans ce chapitre qu'en utilisant les concepts proposés par IDM, nous sommes en mesure de formuler de façon générique le problème d'initialisation et d'observation de modèles de SES en mettant en œuvre des DSL. Pour être en mesure d'utiliser IDM, nous devons formuler le processus d'initialisation et d'observation en termes d'IDM (Figure 50) en spécifiant, d'une part pour le processus d'initialisation, les méta-modèles pour décrire les sources de données des thématiciens et leurs structures (L1), les structures de données du modèle de simulation (L3), la

chaîne de transformations entre les données externes et le modèle de simulation (L2); d'autre part pour le processus d'observation, les méta-modèles pour décrire des sous-parties de trajectoires du modèle de simulation que l'on souhaite observer (les observables - L4), la structure des indicateurs que l'on souhaite construire (L6) et la transformation entre les observables et les indicateurs (L5).

Afin de faire une validation d'usage et montrer l'intérêt de l'approche aux thématiciens, nous allons considérer le modèle de simulation ECEC [Pepper et Smuts 2000] comme « fil rouge » pour la suite, afin d'illustrer l'approche utilisée et afin de montrer le processus complet d'initialisation et d'observation que nous proposons [Rakotonirainy et al. 2014].

Chapitre V. Initialisation de modèles de SES

5.1. Introduction

L'initialisation d'un modèle de simulation consiste tout d'abord à décrire les sources de données dont on a besoin, à spécifier leurs structures pour savoir de quoi parlent-elles; ensuite à décrire le modèle de simulation sous forme de spécification partielle d'une fonction de temps ; et enfin à spécifier une chaîne de transformations des structures de données spécifiées vers celles du modèle de simulation afin d'instancier et alimenter les différents éléments du modèle de simulation par les données externes disponibles.

Dans ce chapitre, nous présentons la mise en œuvre de trois DSL (L1, L2 et L3) permettant de décrire avec souplesse les sources de données avec leurs structures, les structures de données du modèle de simulation ainsi que la chaîne de transformations entre elles.

Pour être générique, nous considérons que la construction de l'état initial du modèle de simulation, la spécification de séries temporelles (par exemple, les séries climatiques) et le paramétrage des processus constituent un seul élément, à savoir la spécification partielle d'une fonction temporelle. En effet, quelque soit le type de modèle, on peut considérer qu'un modèle de simulation est de la forme $\frac{dx}{dt} = f(x)$, où x est l'état de tout type de modèle à un instant t de la simulation et t peut être discret ou continu. Par conséquent, une simulation peut être considérée comme la construction de la fonction $\frac{dx}{dt} = f(x)$, définie du temps initial au temps final dont l'initialisation en est une spécification partielle. L'état initial est la valeur de cette fonction au temps initial, une série temporelle est une sous-partie de cette fonction et le paramétrage spécifie des fonctions du temps sous différentes formes comme les équations différentielles, les équations aux différences ou les interpolations linéaires.

Nous avons montré qu'en utilisant les concepts proposés par IDM, nous sommes en mesure de formuler de façon générique le problème d'initialisation des modèles de SES. La Figure 59 qui est une partie de la Figure 50 montre la correspondance entre la pyramide de modélisation de l'OMG et le processus d'initialisation proposé.

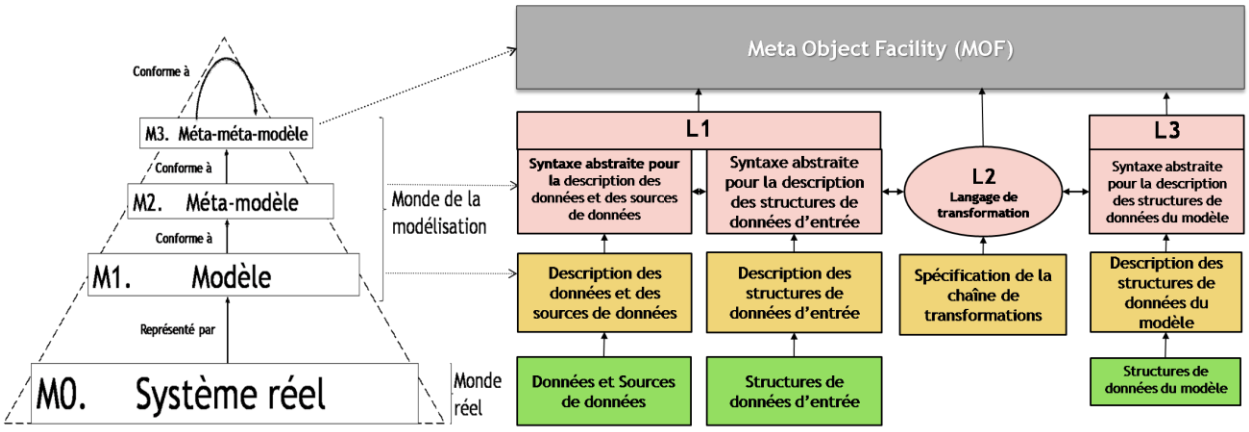


Figure 59. Correspondance entre la pyramide de modélisation de l'OMG et le processus d'initialisation [Rakotonirainy et al. 2015a]

La Figure 60 présente les étapes que nous proposons pour initialiser les modèles de SES à partir des données à la disposition des thématiciens. En effet, après avoir spécifié les sources de données hétérogènes issues des thématiciens (à l'aide de L1), deux possibilités peuvent se présenter pour chaque source de données : dans le cas où elle dispose d'un schéma de ces données (métadonnées), nous avons mis en place un outil permettant d'extraire automatiquement cette métadonnée pour construire la structure de données intermédiaire indépendante de la source de données ; sinon, il sera nécessaire pour l'utilisateur de la spécifier manuellement. Ensuite, cette structure de données intermédiaire va être transformée (à l'aide de L2) en une autre structure de données conforme à celle utilisée dans le modèle de simulation et qui considère le modèle de simulation comme une spécification partielle de la fonction du temps (à l'aide de L3). Enfin, cette série de transformations permet de construire et d'initialiser le modèle de simulation à partir des données stockées dans les différentes sources de données hétérogènes.

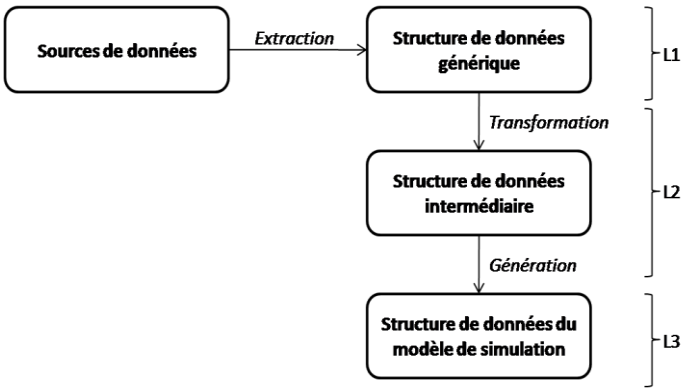


Figure 60. Processus d'initialisation de modèle

5.2. Mise en œuvre des langages L1, L2 et L3

L'objectif est de définir l'initialisation indépendamment des sources et de la manière dont le modèle de simulation est programmé. Pour spécifier le processus d'initialisation, nous proposons de mettre en œuvre les trois DSL suivants :

- Le langage (L1) pour décrire le contenu des sources de données indépendamment des sources hétérogènes ;
- Le langage (L3) pour décrire le modèle de simulation indépendamment de la plateforme de M&S sous la forme d'une spécification partielle d'une fonction de temps ;
- Le langage de transformation (L2) pour mettre en correspondance les structures de données générées à partir du langage L1 et décrites par L3 afin d'initialiser le modèle de simulation.

5.2.1. Langage L1

Le langage L1est un DSL permettant de décrire les différentes sources de données ainsi que les données disponibles, puis de générer ou de spécifier éventuellement les structures de données intermédiaires correspondant aux sources de données qui vont servir par la suite de structures de données d'entrée du modèle de simulation.

5.2.1.1. Principe et état de l'art

Pour le langage L1, nous avons besoin d'un méta-modèle qui permet de spécifier toute source de données, et de spécifier les structures de données à engendrer indépendamment de ces sources. Par rapport à ces besoins, d'une part, les outils ETL (Extract - Transform -Load) comme Informatica, Talend Open Studio, GeoKettle, Pentaho Data Integration, CloverETL permettent de spécifier des sources de données à des tiers, puis d'extraire, de transformer et de générer automatiquement des données à partir de cette spécification. D'autre part, le projet Geotools qui est une boîte à outils SIG libre et gratuite développée en Java permet aussi d'accéder et d'extraire des schémas de données géo-référencées.

Par ailleurs, par rapport à l'exploitation des données issues des sources de données pour initialiser le modèle de simulation, Gio W. a proposé d'une part une architecture d'intégration de données appelée « architecture médiateur » ou « intégration virtuelle » [Gio 1992]. Le principe consiste à concevoir un médiateur qui fournit un schéma global et un vocabulaire unique pour l'expression

des requêtes des utilisateurs. Des auteurs ([Widom 1995], [Chrisment et al. 2005]) ont proposé d'autre part l'architecture « entrepôt de données » ou « intégration matérialisée ». Si dans l'intégration virtuelle, les données restent stockées dans les sources d'origine, dans l'intégration matérialisée, les données sont extraites des sources d'origine et stockées dans un entrepôt de données. Nous adopterons le principe d'intégration virtuelle car nous considérons que les données resteront toujours dans leurs sources de données d'origine et ce sont les structures de données intermédiaires générées ou spécifiées à partir de L1 qui permettront de les extraire au fur et à mesure, lors de l'initialisation du modèle de simulation.

Notre cahier des charges a été donc largement inspiré par ces approches et sera mis en œuvre en utilisant EMF (Eclipse Modeling Framework) [Steinberget al. 2008]. La différence réside dans la génération de structures de données, plutôt que de données.

5.2.1.2. Méta-modèle de L1

Nous avons défini un méta-modèle (Figure 61), dont:

1. Une partie est utilisée pour décrire les sources de données hétérogènes. Elles peuvent être des bases de données, des fichiers Excel, des shapefiles, des CSV et même des formulaires de saisie d'une interface graphique, donc finalement toute sorte de supports et d'applications qui fournissent des données.
2. Une autre partie est utilisée pour spécifier une structure de données intermédiaire construite à partir des diverses sources de données. Car si les données sont stockées ou saisies sur des supports divers, leur traitement nécessite de les représenter dans des structures de données afin de comprendre leur signification.

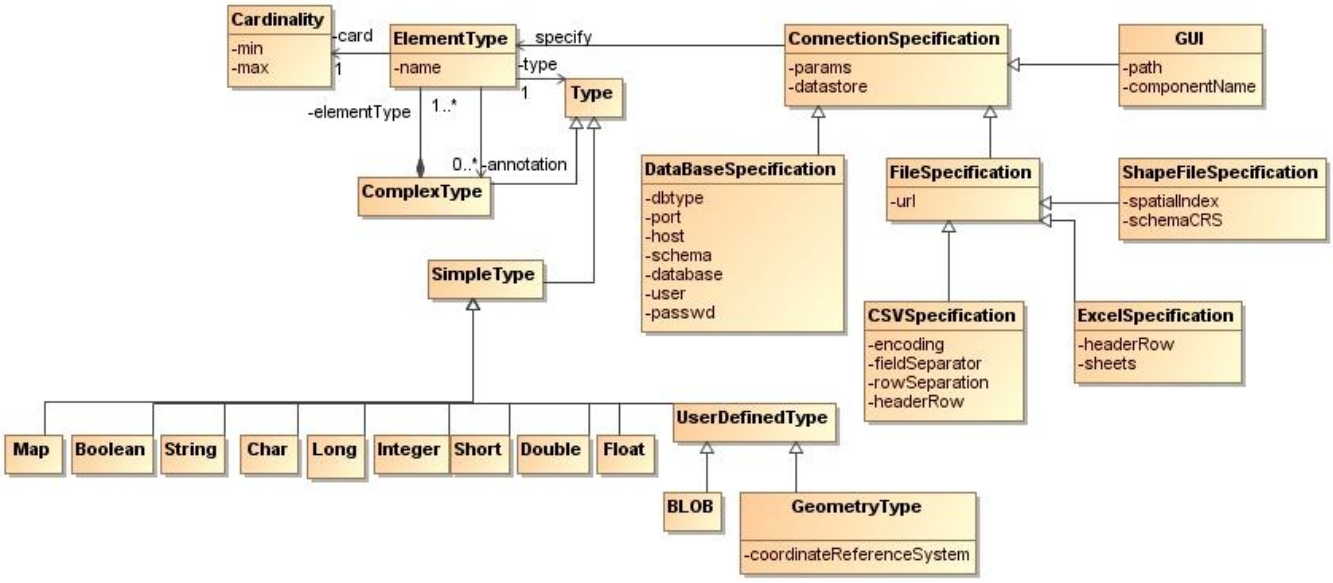


Figure 61. Méta-modèle du langage L1

5.2.1.3. Syntaxe concrète de L1

Pour que le méta-modèle soit utilisable, on peut l'associer à une syntaxe concrète textuelle ou graphique. Afin de construire un modèle conforme à L1, nous avons développé d'une part, une syntaxe concrète textuelle avec le framework XText pour définir une grammaire formelle utilisable par les thématiciens; D'autre part, une syntaxe concrète graphique avec le framework GMF, en plus de la syntaxe concrète arborescente générée automatiquement par les outils d'EMF. La grammaire complète du langage L1 est présentée en *annexe A.1*.

Par conséquent, des éditeurs engendrés par XText et GMF sont fournis aux thématiciens pour préciser les sources de données qu'ils utilisent afin de générer les structures génériques des données. Néanmoins, il est éventuellement nécessaire de spécifier certaines structures de données au cas où les métadonnées n'existent pas.

5.2.1.4. Sémantique comportementale de L1

Le code généré par le langage L1 permet automatiquement d'accéder aux différentes sources de données spécifiées par l'utilisateur et de construire suivant les métadonnées disponibles, les structures de données correspondantes. Ces dernières forment les structures de données d'entrée du modèle de simulation. Le langage Xtend a été utilisé pour développer la sémantique comportementale du langage L1. Un extrait de la spécification du générateur de code qui génère

du code en Java est présenté par le Code 9. Le générateur de code complet est présenté en *annexe B.1*. Actuellement, l'outil que nous avons mis en œuvre génère seulement du code en Java mais il est tout à fait possible d'étendre l'outil afin de générer d'autres codes cibles selon le langage et la plateforme disponible.

```
class DatasourceGenerator implements IGenerator {
    override void doGenerate(Resource resource, IFileSystemAccess fsa) {
        var connectionMetamodel = resource.contents.head as ConnectionMetamodel
        for (ConnectionSpecification specification:connectionMetamodel.connectionSpe
            fsa.generateFile("gen"+"/"+"mde"+"/"+"datasources"+"/"+ //package
            specification.name+".java", specification.genSpecification)
        }
    }
    // Java generator...
    def dispatch genSpecification(DatabaseSpecification connectionSpec)'''
        package gen.mde.datasources;
        import connection.specification.core.DataBaseSpecification;
        import java.io.IOException;
        import connection.specification.core.DataStructureExplorer;
        //Database specification
        public class «connectionSpec.name» extends DataBaseSpecification{
            DataBaseSpecification  «connectionSpec.name.toFirstLower» = new DataBase
            public «connectionSpec.name»(DataStructureExplorer obs)throws IOExceptio
            «connectionSpec.name.toFirstLower».setDbType("«connectionSpec.dbtype»");
            «connectionSpec.name.toFirstLower».setDataBase("«connectionSpec.database
            «connectionSpec.name.toFirstLower».setUser("«connectionSpec.user»");
            «connectionSpec.name.toFirstLower».setPasswd("«connectionSpec.passwd»");
            «connectionSpec.name.toFirstLower».setHost("«connectionSpec.host»");
            «connectionSpec.name.toFirstLower».setPort(«connectionSpec.port»);
            «connectionSpec.name.toFirstLower».setSchema("«connectionSpec.schema»");
            «connectionSpec.name.toFirstLower».getConnection();
            «connectionSpec.name.toFirstLower».database2elementType();
            «connectionSpec.name.toFirstLower».loadDataBaseTree(obs);
            System.out.println("\n");
            }
        }
    '''
    def genMain(ConnectionMetamodel connectionMetamodel)'''
```

Code 9. Extrait du code XTend utilisé par le langage L1 pour générer du code en Java

5.2.1.5. Illustration du langage L1 sur le modèle ECEC

- **Description des sources de données et des structures de données d'entrée du modèle ECEC**

En utilisant le langage L1, nous pouvons décrire les sources de données dont nous disposons afin de construire les structures de données d'entrée du modèle ECEC [Pepper et Smuts 2000]. Le Code 10 et la Figure 62 montrent respectivement, l'utilisation des syntaxes concrètes textuelles

(développée avec XText) et graphiques du langage L1 (développée avec GMF) pour décrire les sources de données afin de générer le code permettant de créer les structures de données indépendamment de ces sources de données utilisées. Le code généré à partir du langage L1 permet de lire les schémas des sources de données spécifiées tels que :

- Le fichier CSV (dénoté par *EcecAgents*) qui décrit le nombre des agents de type « Prédateur » ;
- Le fichier Excel (dénoté par *LogisticParameters*) qui décrit les paramètres de la fonction logistique pour le processus de croissance des plantes de type « Ecology » ;
- La base de données PostgreSQL (dénoté par *Metabolisms*) pour paramétrer les caractéristiques des agents de type « Prédateur » ;
- Un autre fichier Excel (dénoté par *EcecParameters*) pour paramétrer les autres éléments nécessaires pour le fonctionnement du modèle de simulation dans MIMOSA.

```
DatasourceSpecification{
    CSVSpecification "EcecAgents"{
        url: "C:/Users/Hasina/ECEC.csv"
        headerRow: true
    }
    ExcelSpecification "LogisticParameters"{
        url: "C:/Users/Hasina/documents/parameters.xls"
        headerRow: true
        sheet:"logistic"
    }
    DatabaseSpecification "Metabolisms"{
        dbtype:"postgis"
        user : "ecec"
        passwd : "ecec"
        host : "localhost"
        port : "5432"
        schema:"public"
        database: "Forager"
    }
    ExcelSpecification "EcecParameters"{
        url: "D:/Projets/Ezaka2/ecec/input/ececParam.xls"
        headerRow: true
        sheet: "all"
    }
```

Code 10. Spécification des sources de données du modèle ECEC avec la syntaxe concrète textuelle de L1

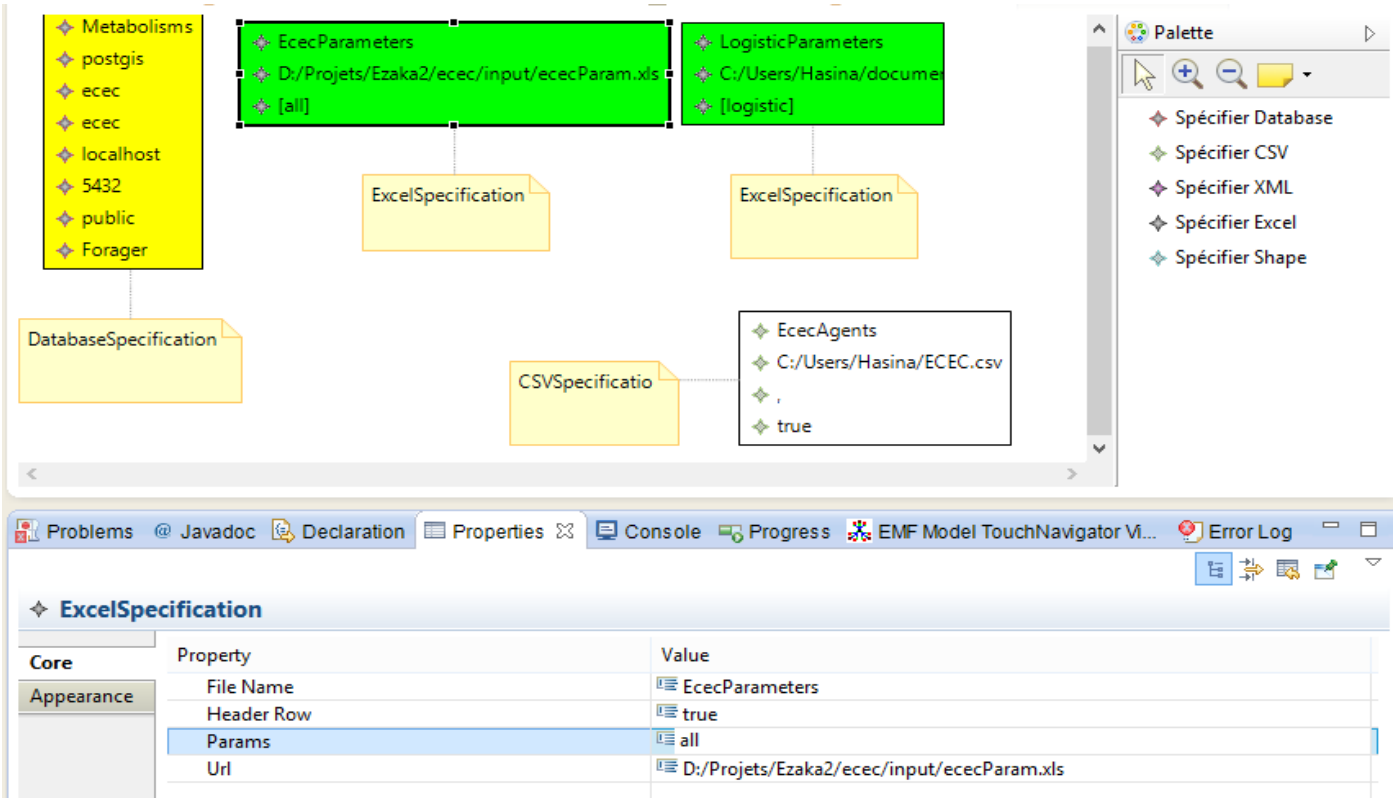
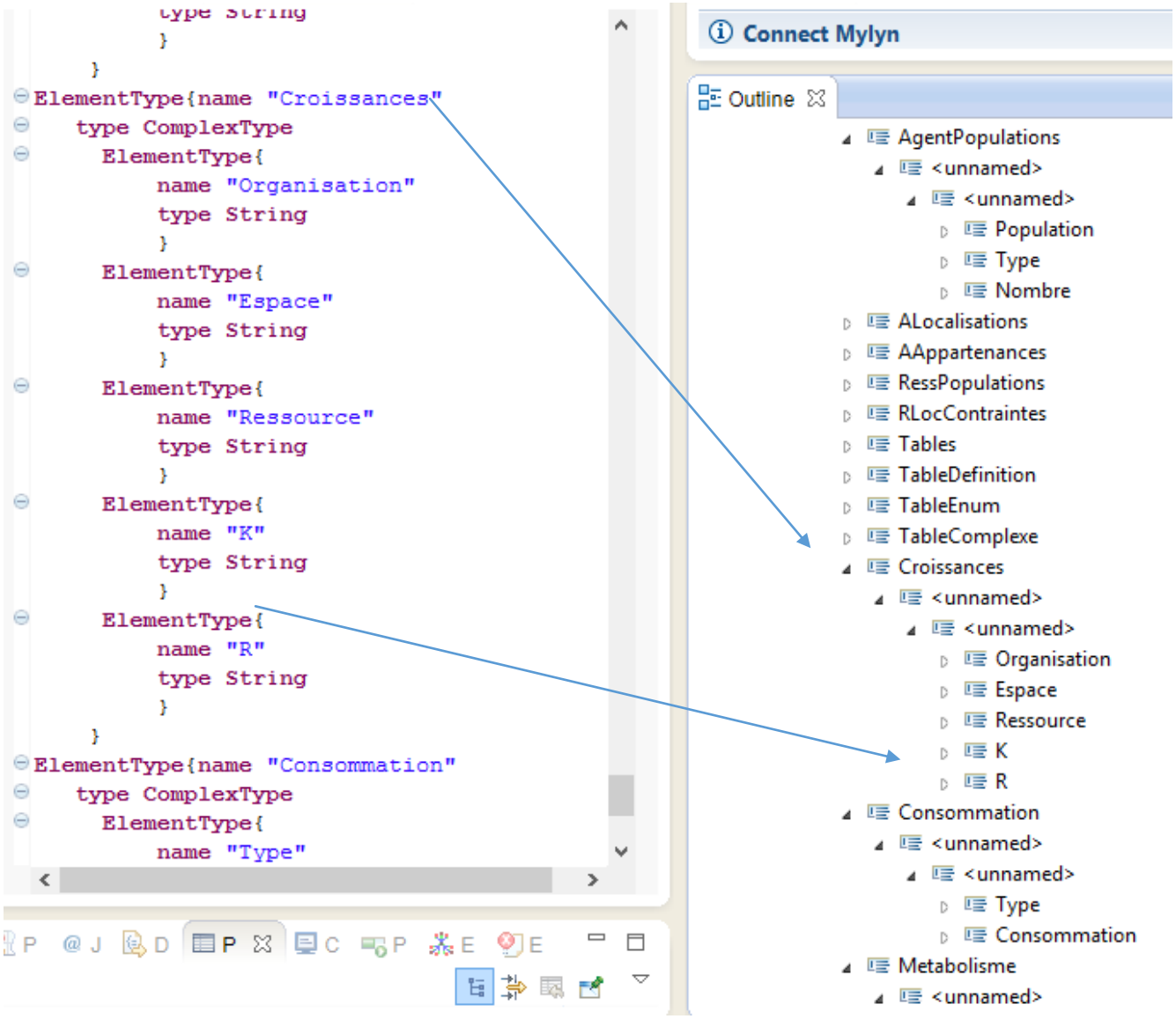


Figure 62. Spécification des sources de données du modèle ECEC avec la syntaxe concrète graphique de L1

Grâce au générateur de code que nous avons développé (Code 9), la description des sources de données génère automatiquement le code en Java qui permet de construire les structures de données intermédiaires (Code 11) issues des sources de données décrites par l'utilisateur.



Code 11. Extrait de la structure de données générée à partir de la spécification des sources de données du modèle ECEC sous forme textuelle et arborescente

5.2.2 Langage L3

Le langage L3 est un DSL utilisé pour décrire le modèle de simulation, indépendamment de la plateforme de M&S.

5.2.2.1. Principe de L3

L'initialisation ne spécifie pas seulement l'état du système au temps initial mais aussi le paramétrage des processus, notamment, la spécification partielle de certaines séries temporelles (par exemple, des séries climatiques), voire des équations explicites ou différentielles. En conséquence, la cible n'est pas qu'un état représenté par une structure de données, mais doit être

considérée comme une fonction du temps que nous allons spécifier partiellement. Il nous paraît donc important de considérer cette fonction totale comme un objet théorique partiellement défini à l'initialisation et complété par la simulation. Formellement, nous posons donc la fonction $\Sigma : [t_{initial},t_{final}] \rightarrow S$ où S est l'ensemble des états possibles du système. Cette fonction est aussi appelée trajectoire dans [Zeigler et al. 2000]. L'état initial devient la spécification de $\Sigma(t_{initial})$. Une série temporelle ou le paramétrage deviennent la spécification d'une partie de Σ.

5.2.2.2. Méta-modèle de L3

En utilisant Ecore, le langage L3 que nous proposons permet de décrire le modèle de simulation comme étant une fonction du temps. Plusieurs types de fonctions sont possibles, à savoir :

- $\Sigma(t) = s$, telle que la valeur est la même pour tout temps ;
- $\Sigma = f(t)$, où f est une fonction explicite du temps (par exemple, a*t+b) ;
- $\Sigma = df(t)$, où df est une équation différentielle simple du temps ;
- $\Sigma = F(t)$ qui définit une équation aux différences ;
- $\Sigma = step((t_1,s_1),\ldots,(t_n,s_n))$, qui définit une fonction constante par palier telle que $t_1 \geq t_{initial}$, $t_n \leq t_{final}$ et on considère que $\Sigma(t) = s_1$ dans l'intervalle $[t_{initial},t_2[$, si dans l'intervalle $[t_i, t_{i+1}[$ et sn dans l'intervalle $[t_n,t_{final}]$;
- $\Sigma = I((t_1,s_1),\ldots,(t_n,s_n))$ définit une interpolation linéaire ou d'autre modèle d'interpolation plus complexe, tel que $t_1 \geq t_{initial}$, $t_n \leq t_{final}$.

Ces fonctions du temps peuvent être spécifiées de différentes manières en fonction de leur nature. Le méta-modèle que nous proposons permet de spécifier:

- Des valeurs (pour paramétrer les fonctions constantes) ;
- Un ensemble de paramètres (accessoirement pour paramétrer les fonctions explicites du temps et les équations différentielles) ;
- Un ensemble de paramètres et un pas de temps (accessoirement pour paramétrer les équations aux différences) ;
- Une séquence de valeurs, une date de début et un pas de temps (accessoirement pour paramétrer les fonctions constantes par palier) ;
- Une liste de couples temps/valeur avec le type d'interpolation (accessoirement pour paramétrer une interpolation linéaire ou d'autres modèles) ;

Les paramètres ou les valeurs ainsi spécifiés peuvent être définis directement à partir de L3, ou provenir des données générées à partir de L1.

Le méta-modèle de L3 est présenté sur la Figure 63.

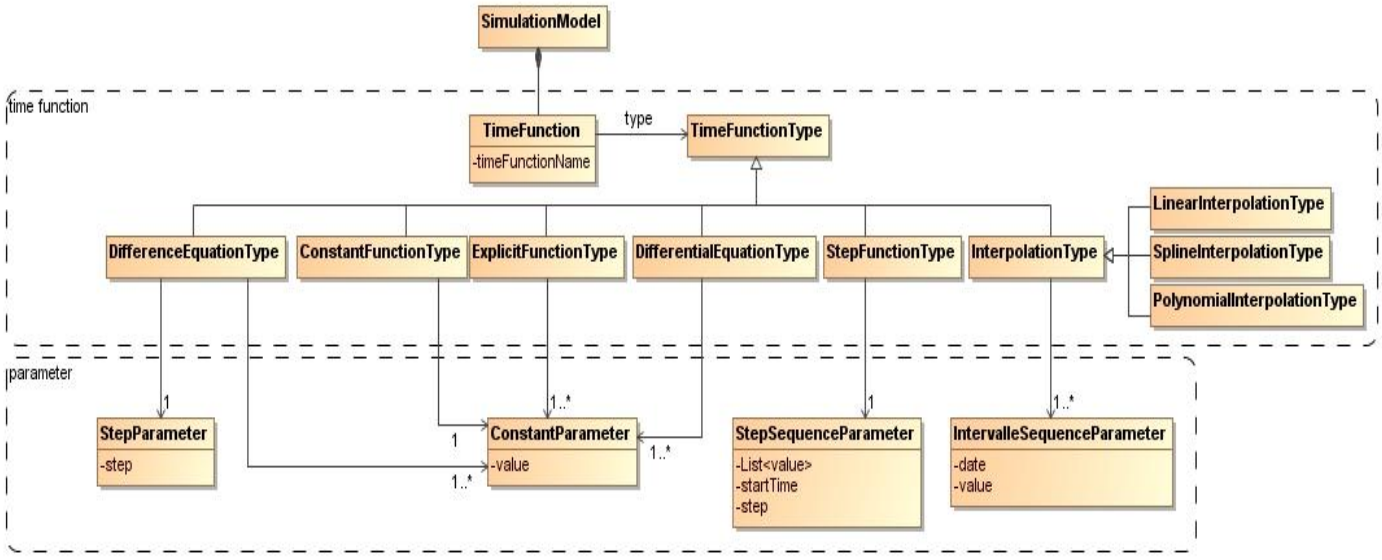


Figure 63. Méta-modèle du langage L3

5.2.2.3. Identification des éléments à initialiser dans le modèle

Pour faciliter la construction de la spécification des fonctions du temps dans le langage L3, nous avons défini une interface graphique (Figure 64) qui permet d'extraire par introspection les différents éléments ou concepts du modèle de simulation qui nous intéresse ainsi que leurs propriétés quel que soit le langage de programmation utilisé pour son implémentation. Il est ensuite nécessaire pour chaque attribut sélectionné de définir une fonction de temps afin d'être conforme au langage L3. Un extrait de la grammaire du langage L3 est présenté par le Code 12. La grammaire complète est décrite en *annexe A.2*.

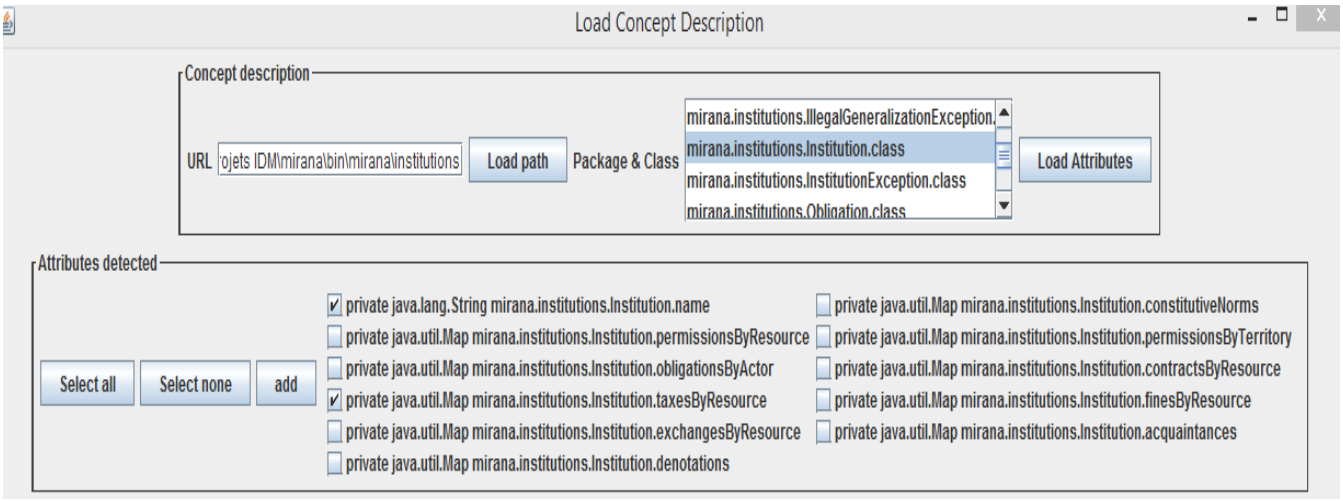


Figure 64. GUI pour extraire les structures de données du modèle de simulation

```
Model:
    {Model}
     concept+=Concept*;

Concept:
    {Concept}
    'Concept' name=STRING
        'represented By' package=STRING 'in' language=Language
        attribute+=Property*
    ;

Property:
    {Property}
    'TimeFunction' functionTimeType=FunctionType'{'
        ('name' name=STRING)?
        ('type' type=STRING)?
    '}'
;

FunctionType:
    dEType = DifferenceEquationType |
    cFType = ConstantFunctionType |
    eFType = ExplicitFunctionType |
    dEqType = DifferentialEquationType |
    iType = InterpolationType
;

InterpolationType:
    {InterpolationType}
```

Code 12. Extrait de la grammaire du langage L3

5.2.2.4. Illustration du langage L3 sur le modèle ECEC

- **Spécification du modèle de simulation**

En utilisant l'interface graphique que nous avons développée, nous sommes en mesure d'extraire toutes les structures de données du modèle ECEC qui ont besoin d'être initialisées (Figure 65) de manière automatique. Une partie de ces structures générée du modèle ECEC est présentée par le Code 13.

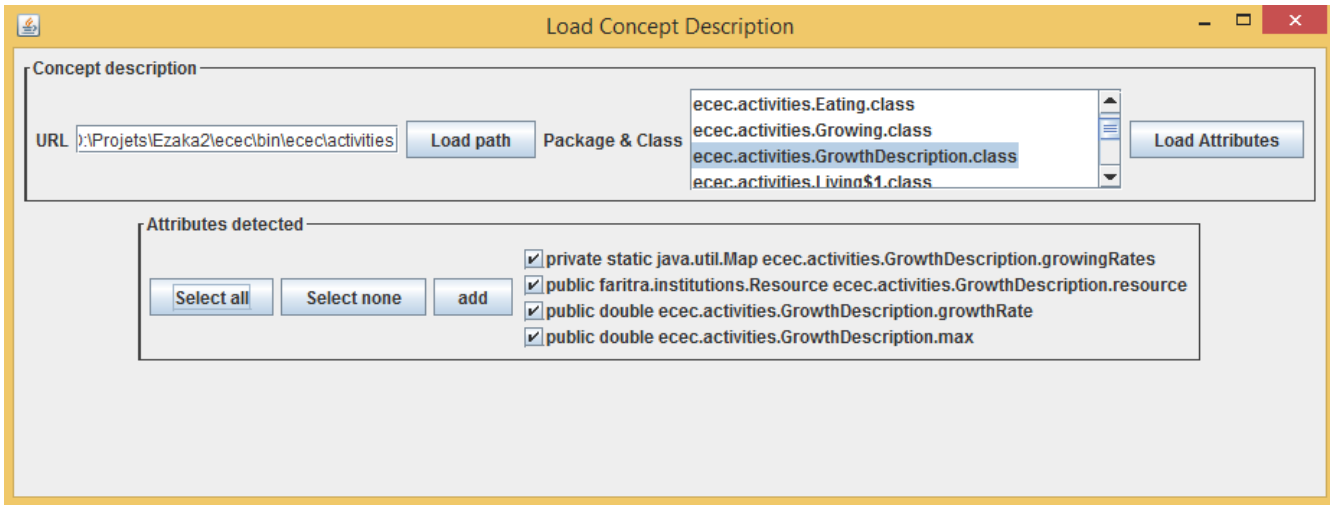
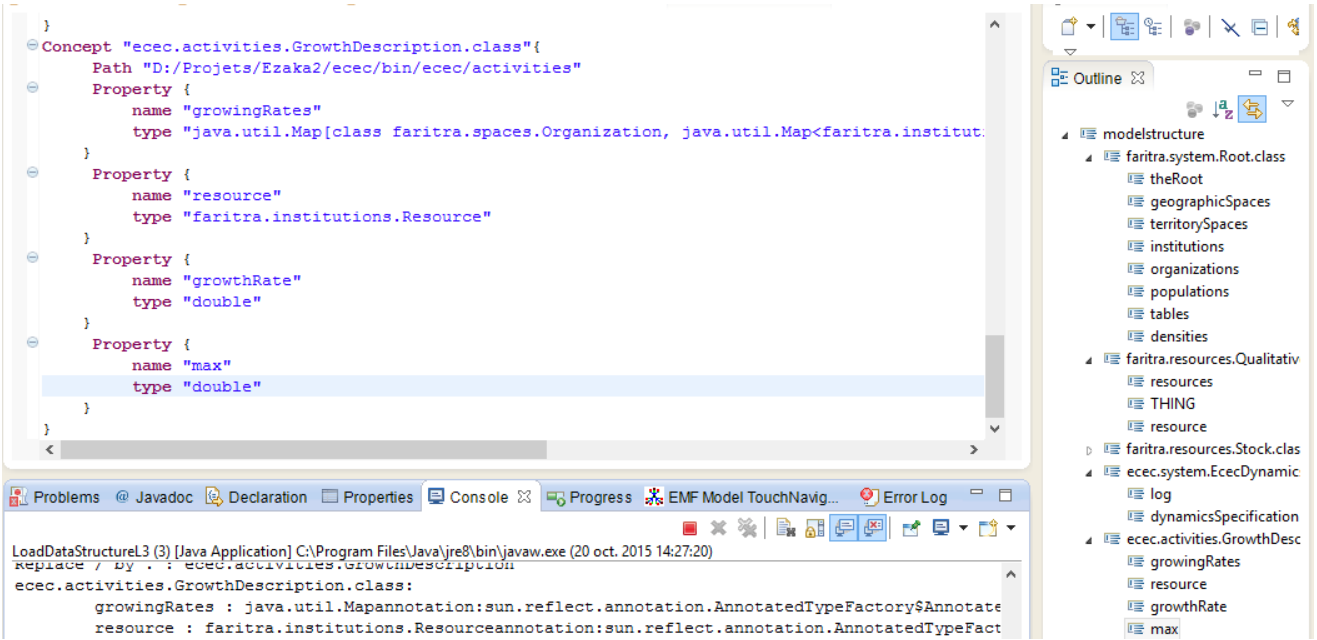


Figure 65. Exemple d'extraction des structures de données de l'activité « Croissance des plantes » du modèle de simulation ECEC



Code 13. Extrait de la structure du modèle de simulation générée à partir de l'interface graphique

5.2.3. Langage L2

L2 est un langage de transformation. Il a pour but de fournir au modélisateur un outil simple à manipuler, capable de transformer les structures de données obtenues à partir du langage L1, pou alimenter les concepts du modèle de simulation spécifiés à partir de L3 sous la forme de fonctions du temps.

5.2.3.1. Principe de L2

Rappelons que le principe fondamental des transformations de modèles est de partir d'un modèle M_a conforme à un méta-modèle source MM_a, puis de spécifier les liens possibles entre le méta-modèle source et le méta-modèle cible pour générer un autre modèle M_b conforme au méta-modèle cible MM_b, tel que MM_a peut être égal ou non à MM_b.

Dans la littérature, il existe plusieurs types d'outils permettant de faire des mises en correspondance, de transformations, de tissages ou de compositions de modèles comme QVT, QVTO, ATL, AML, AMW, Kermeta ou Kompose [El Hamlaoui 2012]. Il est possible d'utiliser un de ces langages génériques de transformation pour générer des modèles conformes à L3 à partir des modèles conformes à L1. Mais, ces langages sont trop génériques et difficiles à prendre en main. Nous proposons alors, de fournir un langage plus spécifique donc plus simple et adapté aux besoins des thématiciens leur permettant de spécifier seulement les transformations possibles pour pouvoir initialiser et paramétrer un modèle de simulation.

En effet, dans la chaîne de transformation de modèles que nous développons:

- Le langage L1 est le méta-modèle source et le langage L3, le méta-modèle cible ;
- Une partie du modèle cible peut être introduite comme modèle source (Figure 66) ;
- Les entrées des transformations sont essentiellement spécifiées par des chemins qui peuvent être des chemins permettant d'explorer la structure de données intermédiaire arborescente générée par L1 mais aussi éventuellement des chemins de projet suivis des structures de données qui implémentent le modèle de simulation décrites par L3;
- Les transformations peuvent être en cascade.

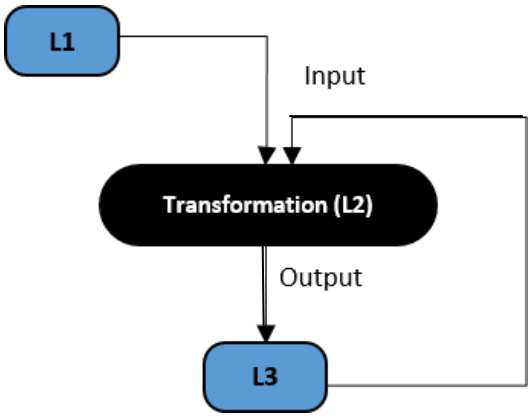


Figure 66. Transformation de L1 vers L3

5.2.3.2. Méta-modèle de L2

Le méta-modèle du langage L2 comprend trois grandes parties (Figure 67), à savoir:

- Une partie « ***input*** » pour spécifier les structures de données d'entrées nécessaires à la transformation. Ces structures de données sont issues des méta-modèles des langages L1 et L3 (en tant que sources) ;
- Une partie « ***transformation*** » pour décrire toutes les transformations possibles des structures de données issues de L1 et de L3 afin de créer des nouvelles structures de données et d'initialiser et paramétrer le modèle de simulation ;
- Une partie **« *output*** » pour spécifier le type de sortie conforme au méta-modèle du langage L3 (le méta-modèle cible) après transformation.

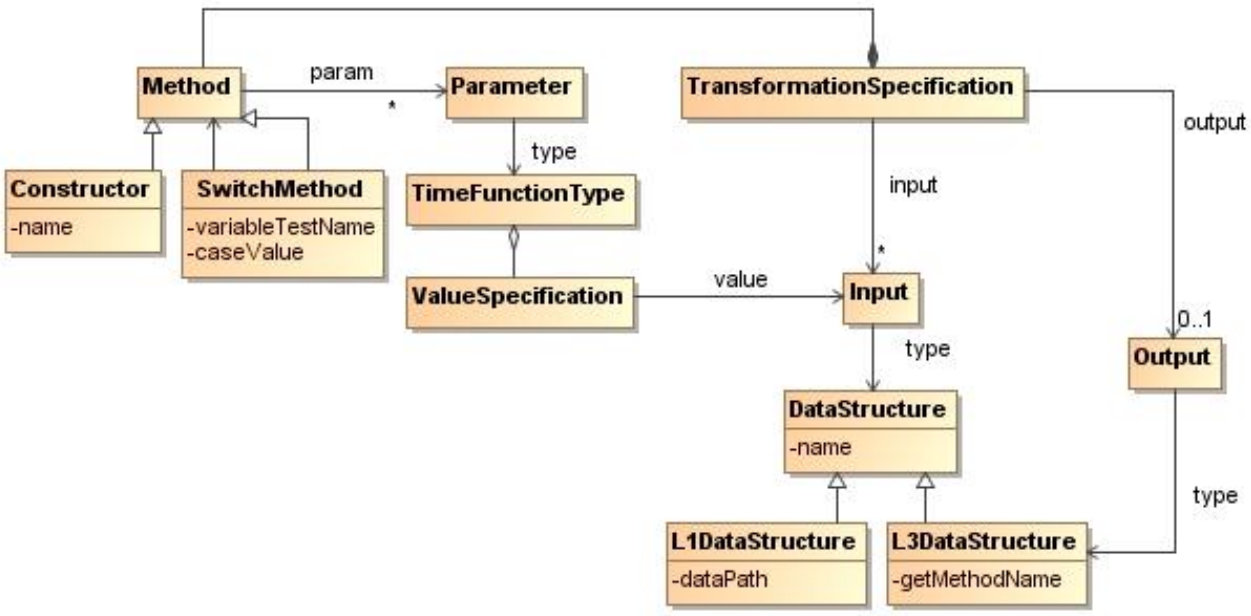


Figure 67.Méta-modèle de L2

5.2.3.3. Syntaxe concrète de L2

La construction d'un modèle de transformation est illustrée par la Figure 68. Le modèle peut être spécifié à partir d'une syntaxe concrète graphique (à gauche), ou textuelle (à droite) ou les deux à la fois. Il est possible ensuite d'avoir des blocs de transformation en cascade, des blocs liés les uns avec les autres par des entrées et des sorties, pour former un graphe non cyclique jusqu'à obtenir le système entier.

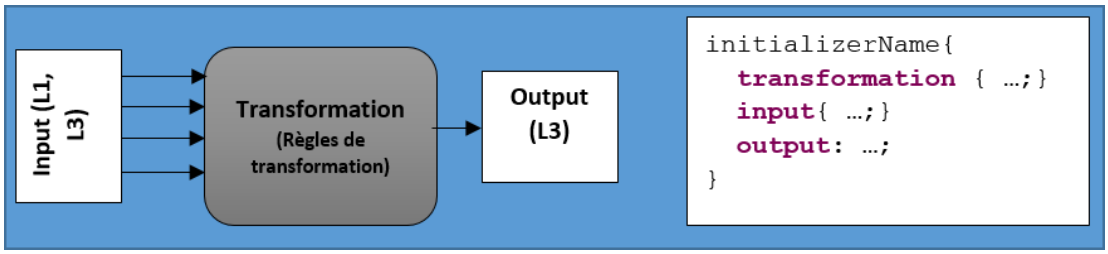


Figure 68. Les trois parties de la syntaxe concrète de L2

Dans un premier temps, nous avons implémenté la grammaire du langage L2 avec XText qui est présentée en *annexe A.3*.

Un programme en langage L2 est structuré de la manière suivante :

- Une partie en-tête (Code 14), qui contient le nom de l'initialiseur avec la liste des paquets nécessaires aux transformations;

```
InitializerName{
package … ;
        // Liste des packages dont on a besoin pour la transformation
}
```

Code 14. En-tête d'un programme en langage L2

- Le corps du programme (Code 15), divisé en trois blocs :

```
transformation {
        // Les méthodes nécessaires permettant la transformation
}
input {
// Les structures de données d'entrée nécessaires pour les méthodes de
transformation
}
output {
// La sortie obtenue après transformation
        report ;
}
```

Code 15. Corps d'un programme en langage L2

Cette grammaire permet notamment à l'utilisateur de :

- Construire et initialiser le modèle à partir des données externes (L1);
- Spécifier comment obtenir les valeurs des paramètres des fonctions de temps qui ont été spécifiées en L3. En effet, ces valeurs sont obtenues à partir de la description des chemins qui mènent aux structures de données correspondantes aux valeurs.
    - Pour une fonction constante ($\Sigma(t) = s$), nous avons besoin de spécifier un chemin permettant d'extraire un seul paramètre ;
    - Pour une fonction explicite du temps ($\Sigma = f(t)$) ou une équation différentielle simple du temps ($\Sigma = df(t)$), nous avons besoin de spécifier un chemin permettant d'extraire une séquence de valeurs à pas fixe ;
    - Pour une fonction constante par palier ($\Sigma = step((t_1,s_1),\ldots,(t_n,s_n))$) ou une interpolation linéaire ($\Sigma = linear((t_1,s_1),\ldots,(t_n,s_n))$ telle que $t_1 \geq t_{initial}$, $t_n \leq t_{final}$), nous devons spécifier un chemin permettant d'extraire une séquence de valeurs avec intervalles.

Toutes ces spécifications sont des descriptions indépendantes d'autres parties du système.

5.2.3.4. Sémantique comportementale de L2

Le code généré à partir du langage L2 permet, en tenant compte de la technologie utilisée, de mettre en correspondance les différentes structures de données de L1 et de L3 selon la spécification réalisée par l'utilisateur. Cette mise en correspondance permet d'utiliser les données issues des différentes sources de données pour alimenter le modèle de simulation. Le langage Xtend a été utilisé pour développer le générateur de code de L2. Un extrait de la spécification de ce générateur de code est présenté par le Code 16. Le générateur de code complet est présenté en *annexe B.2*.

```
class L2Generator implements IGenerator {

    override void doGenerate(Resource resource, IFileSystemAccess fsa) {
     var modelInit = resource.contents.head as ModelInit
        fsa.generateFile("gen"+"/"+"mde"+"/"+"run"+"/"+ //package
        modelInit.initializationName.toFirstUpper+".java", modelInit.genCodeJava)
    }

    def genCodeJava(ModelInit modelInit)'''
    def genMainMethodJava(ModelInit modelInit) '''
    public static void main(String [] args) throws BiffException, IOException, Except
        new «modelInit.initializationName.toFirstUpper» (new LoadDataStructureL1().ge
            «FOR initializer:modelInit.initializers»
                «initializer.nameInitializer»();
                «IF initializer.output!=null»
                    System.out.println(«initializer.output.nameOutput»);
                «ELSE»System.out.println("Without output");
                «ENDIF»
            «ENDFOR»
        }
    '''
    def genMethodJava(Initializer init)'''
    public static void «init.nameInitializer»()throws Exception{
        «FOR input: init.input»
            «genInput(input)»
        «ENDFOR»
        «genTransformation(init)»
        }
    '''

    def genTransformation(Initializer init)'''
    «IF init.transformation.method!=null && init.transformation.switchCase==null»
```

Code 16. Extrait du code XTend utilisé par le langage L2 pour générer du code en Java afin d'initialiser un modèle de simulation

5.2.3.5. Illustration de l'utilisation du langage L2 sur le modèle ECEC

Avec le langage L2, on peut transformer les structures de données issues du langage L1 pour pouvoir initialiser le modèle ECEC.

On va donner quelques exemples d'utilisation du langage L2 sur le modèle ECEC afin de montrer comment initialiser les éléments du modèle de simulation.

- **Création d'institutions dans le modèle ECEC**

Etant implémenté dans la plateforme MIMOSA, le modèle ECEC est construit sur la base du modèle générique « *Faritra* ». Comme le modèle Mirana [Aubert et al. 2010], il utilise également la notion d'institution qui est un ensemble de normes, constitutives pour le vocabulaire, régulatrices pour les « lois » s'il y en a. Pour créer des éléments comme des institutions dans le modèle de simulation, il est nécessaire de spécifier l'endroit où se trouve la méthode permettant de les construire, son initialiseur ou son constructeur, puis, de spécifier les données d'entrée nécessaires pour sa construction (Code 17). C'est à la charge du générateur du langage L1 ensuite de chercher le constructeur dans le code source du modèle de simulation ainsi que les données à partir de la structure générique instanciée par L1 pour pouvoir construire des éléments de type « Institution ». En générant les données à partir des structures de données d'entrée, nous avons recensé deux types d'institution, à savoir : « InstitutionEcology » et « InstitutionForager».

```
InstitutionInitializer{
    package "faritra.institutions";
    transformation{
        new Institution(institutionName) ;
    }
    input{
        String institutionName = datalist("L1.ecec1.Institutions.Institution");
    }
    output{
        report institutions;
    }
}
```

Code 17. Construction des institutions du modèle ECEC avec L2

- **Construction des normes constitutives dans le modèle ECEC**

Les normes constitutives définissent la terminologie utilisée dans ECEC sous la forme d'un ensemble de normes structurées par des relations. Les noms utilisés peuvent dénoter soit des objets (les identifiants), soit des ensembles d'objets (les concepts ou catégories). Pour chaque nom, on spécifie quel type d'objet il peut désigner. On distingue ainsi :

- Les rôles qui désignent des individus considérés comme parties d'organisation (par exemple, *écologiste* dans l'institution « InstitutionEcology » et *prédateur* dans l'institution « InstitutionForager ») ;
- Les ressources qui désignent des stocks (par exemple, *biomasse* dans l'institution « InstitutionEcology », *énergie* et *biomasse* dans l'institution « InstitutionForager ») ;

- Les lieux qui désignent des territoires ou parties de territoire (par exemple, *habitat* dans l'institution « InstitutionEcology », *champs* dans l'institution « InstitutionForager »);
- Les actions qui désignent des activités (par exemple, *se croître* dans l'institution « InstitutionEcology » et *se déplacer*, *manger*, *vivre* dans l'institution « InstitutionForager »).

La spécification avec L2 décrite par le Code 18 permet de générer le code permettant de définir les normes constitutives de chaque institution.

```
ConceptInitializer{
    package "faritra.institutions";
    package "faritra.system";
    transformation{
         switch(role){
            Institution institution = Root.getInstitution(institutionName) ;
            case "lieu": institution.addPlace(concept) ;
            case "acteur": institution.addActor(concept) ;
            case "action": institution.addActivity(concept) ;
            case "ressource": institution.addResource(concept) ;
         }
        }
    input{
      String institutionName = datalist("L1.ecec1.Institutions.Institution");
      String concept = datalist("L1.ecec1.Institutions.Concept");
      String role = datalist("L1.ecec1.Institutions.Categorie");
    }
    output{
        report;
    }
}
```

Code 18. Extrait de la spécification des normes constitutives avec L2

- **Paramétrage de la hiérarchie des concepts utilisés dans le modèle ECEC**

Toujours dans la construction des normes constitutives, nous avons aussi besoin d'initialiser deux types de « Forager » qui sont : « restrained » et « unrestrained ». Avec le langage L2, il est possible en utilisant la méthode *setGeneralization()*, disponible à partir du modèle de simulation, de spécifier des relations de généralisation ou de spécialisation afin d'identifier les cas particuliers de concepts (Code 19).

```
GeneralizationInitializer{
    package "faritra.institutions";
    package "faritra.system";
    transformation{
        Institution institution = Root.getInstitution(institutionName) ;
        ConstitutiveNorm generalNorm = institution.getConcept(general) ;
        ConstitutiveNorm particularNorm = institution.getConcept(particular) ;
        particularNorm.setGeneralization(generalNorm) ;
        }
    input{
      String institutionName = datalist("L1.ecec1.Generalisation.Institution");
      String general = datalist("L1.ecec1.Generalisation.General");
      String particular = datalist("L1.ecec1.Generalisation.Particulier");
    }
    output{
        report;
    }
}
```

Code 19. Paramétrage de la hiérarchie des concepts

- **Paramétrage des points de vue multiple dans le modèle ECEC**

Pendant la simulation, il se peut que les agents appartenant à différentes institutions utilisent des mots différents pour parler de la même chose, ou utilisent les mêmes mots pour parler de choses différentes. Avec le langage L2, en utilisant la méthode *addTranslation()*, disponible à partir du modèle de simulation, il est possible de spécifier les relations sémantiques entre les concepts utilisés dans différentes institutions (Code 20). Par exemple, il est possible de spécifier que le concept « biomasse » dans l'institution « InstitutionForager » a le même sens que celui utilisé dans l'institution « InstitutionEcology » et inversement.

```
TranslationInitializer{
    package "faritra.institutions";
    package "faritra.system";
    transformation{
        Institution theInstitution1 = Root.getInstitution(institutionName1) ;
        Institution theInstitution2 = Root.getInstitution(institutionName2) ;
        ConstitutiveNorm theConcept1 = theInstitution1.getConcept(concept1) ;
        ConstitutiveNorm theConcept2 = theInstitution2.getConcept(concept2) ;
        theInstitution1.addTranslation(theConcept1 theConcept2) ;
    }
    input{
      String institutionName1 = datalist("L1.ecec1.Traductions.Institution1");
      String institutionName2 = datalist("L1.ecec1.Traductions.Institution2");
      String concept1 = datalist("L1.ecec1.Traductions.Concept1");
      String concept2 = datalist("L1.ecec1.Traductions.Concept2");
    }
    output{
        report;
    }
}
```

Code 20. Traduction d'un concept d'une institution vers une autre

- **Paramétrage des activités dans le modèle ECEC**

Les activités du modèle ECEC sont implémentées en Java. Il est donc nécessaire de faire correspondre les mots utilisés par les thématiciens pour désigner les activités ou les actions (par exemple, *to move*, *to eat*, *to live*, *to grow*) avec leurs implémentations informatiques (par exemple, *ecec.activities.Moving.Java*, *ecec.activities.Eating.Java*, *ecec.activities.Living.Java* et *ecec.activities.Growing.Java*). Le Code 21 montre la façon de paramétrer les activités des agents dans le modèle ECEC en utilisant la méthode *defineDenotation()* de la classe *Institution*, disponible à partir du modèle de simulation.

```
ActivityInitializer{
    package "faritra.institutions";
    package "faritra.system";
    transformation{
        Institution institution = Root.getInstitution(institutionName) ;
        ConstitutiveNorm theActivity = institution.getConcept(activity) ;
        institution.defineDenotation(theActivity implementation) ;
    }
    input{
      String institutionName = datalist("L1.ecec1.IActivites.Institution");
      String activity = datalist("L1.ecec1.IActivites.Activite");
      String implementation = datalist("L1.ecec1.IActivites.Implementation");
    }
    output{
        report;
    }
}
```

Code 21. Paramétrage des activités dans le modèle ECEC

- **Paramétrage du processus de croissance des plantes**

La fonction logistique[41] permet la croissance de la biomasse des plantes pendant la simulation. Elle dépend des valeurs des paramètres stockées dans les sources de données spécifiées par l'utilisateur. Le Code 22 montre la façon dont le processus de croissance des plantes est paramétré avec le langage L2.

```
// Logistic equation
CroissanceEcologyInitializer{
    package "faritra.system";
    package "faritra.table.Tuple";
    transformation{
        Table table = Root.getTable(growthTable) ;
        Organization organization = (Organization) table.get(organisation) ;
        ConstitutiveNorm spaceName = (ConstitutiveNorm) table.get(espace) ;
        Resource resource = (Resource)table.get(ressource) ;
        double rate = table.get(taux) ;
        double max = table.get(capacite) ;
     }
    input{
        String organisation = datalist("L1.ecec1.Croissances.Organisation");
        String espace = datalist("L1.ecec1.Croissances.Espace");
        String ressource = datalist("L1.ecec1.Croissances.Ressource");
        double taux  = datalist("L1.ecec1.Croissances.R");
        double capacite = datalist("L1.ecec1.Croissances.K");
        String growthTable = "GrowthTable";
    }
    output{
        report;
    }
}
```

Code 22. Paramétrage du processus de croissance des plantes

- **Initialisation du nombre d'agents « Forager »**

Le modèle ECEC implémente deux catégories d'agents « Forager », à savoir : Restrained et Unrestrained. L'initialisation du nombre d'agents de chaque catégorie est possible avec L2 en utilisant la méthode *addPopulation()*, disponible à partir du modèle de simulation. Cette méthode permet de créer et d'ajouter un certain nombre d'agents parmi la population du modèle de simulation. Elle est spécifiée par le Code 23.

[41] Fonction logistique : $X_{t+1} = X_t + rX_t\left(1 - \frac{X_t}{K}\right)$

```
// Agents
AgentPopulationsInitializer {
    package "faritra.institutions";
    package "faritra.system";
    transformation{
        Root.addPopulation(name size type) ;

    }
    input{
      String name = datalist("L1.ecec1.AgentPopulations.Population");
      String type = datalist("L1.ecec1.AgentPopulations.Type");
      Integer size = datalist("L1.ecec1.AgentPopulations.Nombre");
    }
    output{
        report;
    }
}
```

Code 23. Initialisation du nombre d'agents de type prédateur

- **Initialisation de la position initiale des agents dans le modèle ECEC**

Avec le langage L2, il est possible d'appeler directement la méthode *initialize()* de la classe *Population* afin d'initialiser la position spatiale et institutionnelle de chaque agent appartenant à la population du modèle de simulation (Code 24).

```
PositionPopulationInitializer{
    package "faritra.system";
    package "faritra.spaces";
    transformation{
        Population population = (Population) Root.getPopulations() ;
        population.initialize() ;
        }
    input{
    }
    output{
        report;
    }
}
```

Code 24. Position spatiale des agents dans le modèle ECEC

- **Générateur de code pour initialiser le modèle de simulation**

La spécification de la chaîne de transformation avec L2 (dont un extrait est présenté par le Code 25) génère automatiquement le code en Java permettant de construire et initialiser le modèle de simulation. L'exécution de ce dernier génère ensuite l'état initial du modèle ECEC (Figure 69).

```
public static void ActivityInitializer()throws Exception{
    List<Object> institutionName = inputList("L1","ecec1","IActivites","Institution");
    System.out.println(institutionName);
    List<Object> activity = inputList("L1","ecec1","IActivites","Activite");
    System.out.println(activity);
    List<Object> implementation = inputList("L1","ecec1","IActivites","Implementation");
    System.out.println(implementation);
    for (int i = 0; i < institutionName.size(); i++){
    // Test if data is null...
    if(!institutionName.get(i).toString().equals("")){
        Institution institution =
        Root.getInstitution
        (
            institutionName.get(i).toString()
            );
        ConstitutiveNorm theActivity =
        institution.getConcept
        (
            activity.get(i).toString()
            );
        institution.defineDenotation
        (
        (ConstitutiveNorm)theActivity,
            implementation.get(i).toString()
            );
        }//end if
    }
```

Code 25. Extrait de code en Java pour le paramétrage des activités des agents du modèle ECEC, généré à partir de L2

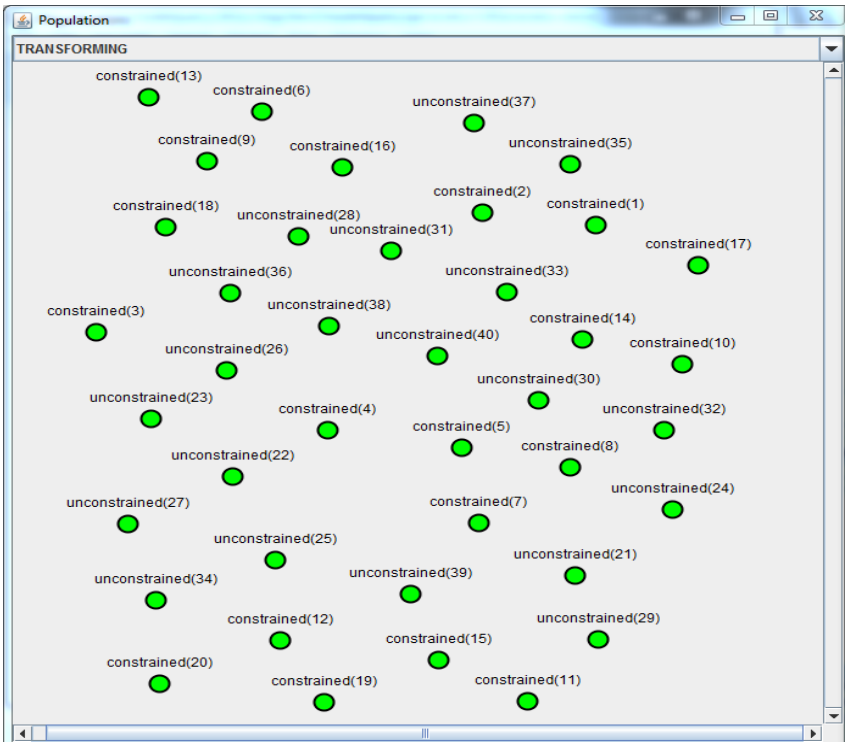


Figure 69. Génération graphique de l'état initial des prédateurs dans l'environnement

5.3. Conclusion

Nous avons montré par la mise en œuvre des langages L1, L2 et L3 que d'une part, nous sommes en mesure d'exploiter les données hétérogènes issues des thématiciens pour paramétrer les modèles de simulation ; d'autre part, on peut formuler de façon générique les problèmes d'initialisation des modèles de SES en utilisant les concepts proposés par l'IDM. On a pris comme exemple d'application, le modèle ECEC [Pepper et Smuts 2000] afin de démontrer et valider la généricité du processus d'initialisation de modèle de simulation proposé. Cette approche a été également appliquée pour faciliter l'initialisation du modèle Mirana [Aubert S. et al. 2010].

En effet, les langages L1, L2 et L3 développés permettent à l'utilisateur, d'abord, de décrire les sources de données hétérogènes dont il dispose, puis d'extraire et/ou de spécifier des structures de données (L1). Cette structure de données va être manipulée pour pouvoir initialiser le modèle de simulation. La mise en œuvre du langage L1 est inspirée des techniques et des outils d'extraction de données comme les outils ETL et les architectures d'intégration de données. La différence réside dans l'extraction de structures de données plutôt que de données. Ensuite, comme l'initialisation ne spécifie pas seulement l'état du système au temps initial mais aussi la spécification partielle de certaines séries temporelles, nous avons considéré le modèle de simulation comme une fonction du temps spécifiée partiellement (L3). Une interface graphique permettant d'extraire automatiquement les différentes structures d'un modèle de simulation est mise à disposition des utilisateurs afin de faciliter la spécification de celles-ci. Enfin, pour pouvoir initialiser le modèle de simulation, nous avons mis à disposition des thématiciens, un langage de transformation (L2). L'élaboration de ce langage de transformation est issue de plusieurs expériences sur des exemples concrets de transformations nécessaires pour construire les structures de données des modèles Mirana et IMAS à partir des bases de données des thématiciens.

Le prototype que nous avons présenté génère du code en Java. Ceci est dû à l'utilisation de la plateforme de M&S MIMOSA comme plateforme cible, mais il est tout à fait possible et envisageable de travailler avec un modèle de simulation (comme ECEC ou Mirana ou autres) implémenté dans d'autres plateformes de M&S comme TerraME, Netlogo, GAMA, Cormas, avec tout type de langage de programmation. La sémantique des outils mis en place est extensible et permet par la suite de générer différents types de codes cibles selon le choix de l'utilisateur, en utilisant les mêmes syntaxes concrètes spécifiées. Ainsi, on a pu mettre à disposition, des outils et un savoir-faire permettant aux thématiciens d'exploiter, facilement et de manière générique, les données dont ils disposent.

Chapitre VI. Observation de modèles de SES

6.1. Introduction

Rappelons que les chercheurs ont besoin d'observer ce qui se passe dans le modèle de simulation pour comprendre le fonctionnement des socio-écosystèmes (SES). L'objectif de ce chapitre est de proposer un cadre générique de spécification de l'observation des modèles de SES afin d'engendrer facilement les indicateurs que les thématiciens veulent suivre pendant la simulation.

Nous avons montré (cf. 4.3) qu'en utilisant les concepts proposés par IDM, nous sommes en mesure de formuler de façon générique le problème d'observation des modèles de SES en créant des DSL (Figure 70– une partie de la Figure 50). Pour être en mesure d'utiliser IDM, nous devons formuler le processus d'observation en termes d'IDM en spécifiant les méta-modèles pour décrire les trajectoires de simulation du modèle que l'on souhaite observer (les observables - L4), la structure des indicateurs que l'on souhaite construire (L6) et la transformation entre les trajectoires et les indicateurs (L5).

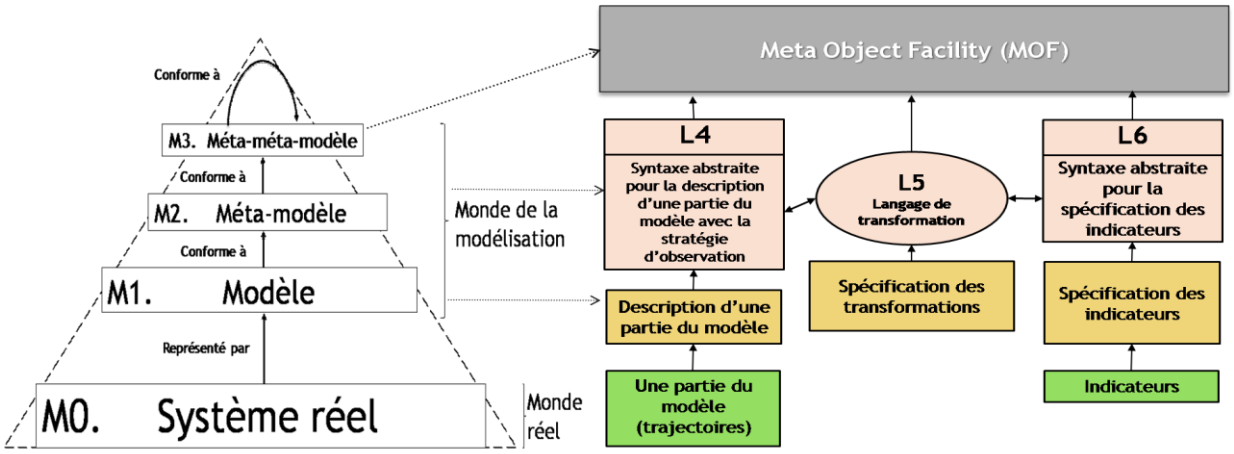


Figure 70. Correspondance entre la pyramide de modélisation de l'OMG et le processus d'observation [Rakotonirainy et al. 2015b, c]

La Figure 71 présente les étapes globales que nous proposons pour construire les indicateurs souhaités par les thématiciens en partant des structures de données du modèle de SES. En effet, pendant la simulation, il est possible de suivre l'évolution des structures de données du modèle en définissant une stratégie d'observation afin d'obtenir des trajectoires. Construire une trajectoire oblige éventuellement à spécifier une stratégie d'échantillonnage. Les trajectoires ainsi obtenues sont des structures de données indexées par le temps (*séquence <dates, structures de données>*). Par conséquent, nous devons encore transformer ces structures de données indexées par le temps

(i.e. les trajectoires) en des structures de données manipulables afin d'obtenir les indicateurs. Les indicateurs peuvent être enfin soit sauvegardés dans un support de données, soit visualisés.

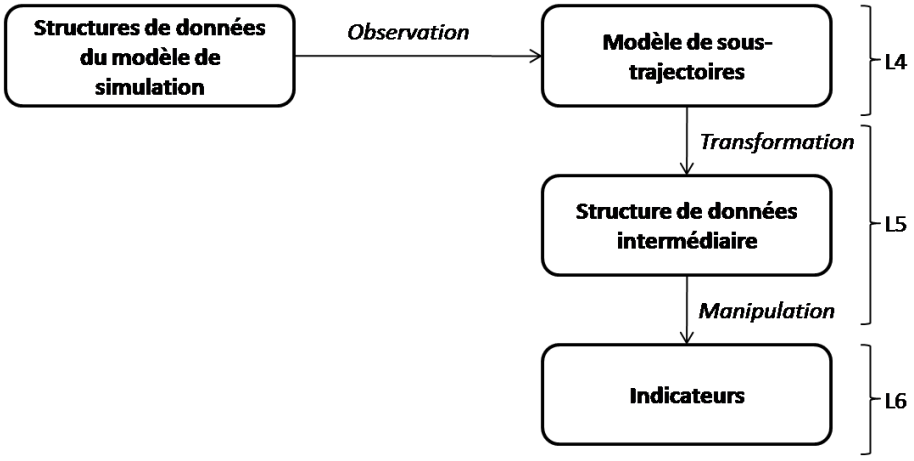


Figure 71.Processus de construction des indicateurs

6.2. Mise en œuvre des langages L4, L5 et L6

Nous avons spécifié les méta-modèles des langages dédiés (DSL) L4, L5 et L6 avec le méta-méta-modèle Ecore du framework EMF (Eclipse Modeling Framework) [Steinberg et al.2008]. Ces DSL permettent de spécifier :

- La partie du modèle que l'on veut suivre pendant la simulation indépendamment de la plateforme de M&S (L4) et comment former un ensemble de trajectoires ;
- Les indicateurs souhaités, indépendamment des puits de données, en charge de faire la présentation visuelle ou le stockage des indicateurs sur des supports de données (L6) ;
- Les transformations des trajectoires en indicateurs (L5).

6.2.1. Le Langage L4

Le langage L4 est conçu pour spécifier la partie du modèle que l'on veut observer et suivre pendant la simulation avec une certaine stratégie d'observation. La solution que nous proposons doit être adaptée aux thématiciens et inspirée de l'évaluation de leurs besoins.

6.2.1.1. Observation de modèles de simulation

Pendant la simulation, les thématiciens ont besoin d'observer des sous-parties du modèle de simulation et non sa totalité, pour calculer les indicateurs qu'ils souhaitent. Observer des sous-

parties d'un modèle de simulation nécessite d'avoir les moyens pour identifier des éléments du modèle de simulation de manière précise au cours de la simulation. Etant donné que la structure de données d'un modèle de simulation est arborescente ou éventuellement un graphe, spécifier un chemin à partir de la racine de celle-ci permet donc d'identifier les éléments que les thématiciens voudraient suivre dans le modèle de simulation. Dans la littérature, des langages de requête comme XPath[42] et XQuery[43] permettent de spécifier des chemins pour parcourir les structures de données arborescentes d'un fichier XML, qui est reconnu comme étant le format de description de données universel [Bray et al. 1998] permettant de représenter la hiérarchie de n'importe quelle structure de données. Ainsi, pour être générique, nous proposons un outil inspiré de ces langages de requêtes afin de spécifier ce qu'on voudrait observer dans la trajectoire du modèle de simulation. Cependant, nous ne travaillons pas sur du XML mais sur les structures de données elles-mêmes.

Le fait de devoir engendrer une trajectoire plutôt qu'un état oblige à ajouter la stratégie d'observation.

6.2.1.2. Stratégies d'observation

Les thématiciens ne veulent pas observer l'évolution de ce qui se passe dans le modèle de simulation de manière continue mais selon des besoins d'observation (par exemple, seulement au début ou à la fin de la simulation, à un pas de temps constant, à chaque changement d'état d'un élément observable, etc.). Il est donc important de pouvoir définir une stratégie d'observation. Cette stratégie consiste à spécifier la façon, le moment ou la fréquence d'observation du modèle pendant la simulation, afin d'obtenir incrémentalement les éléments de la trajectoire. En effet, deux types de stratégie d'observation peuvent être spécifiés, à savoir :

- **$O_{observer}$:** Inspiré du mode d'observation évènementiel du mécanisme d'observation proposé par Quesnel [Quesnel et al. 2012], chaque élément du modèle va signaler son changement d'état à un observateur abonné en utilisant le modèle de conception ou *pattern* "observateur" [Gammaet al. 1995]; Dans ce cas, le modèle de simulation doit implémenter ce type de *pattern* et il est nécessaire de décrire les fonctions qui permettent à un observateur de s'abonner à l'élément qu'on voudrait suivre et comment le changement d'état est signalé.

[42] http://www.w3.org/TR/xpath/
[43] http://www.w3schools.com/xsl/xquery_intro.asp

- **O_{timed}:** Inspiré du mode d'observation planifié du mécanisme d'observation proposé par Quesnel [Quesnel et al. 2012], un observateur interroge l'état d'un élément observable du modèle en spécifiant les dates d'observation (uniques, à pas de temps constant, etc.). Il est alors nécessaire de spécifier comment extraire l'information (*getters*).

Les éléments du modèle seront ensuite extraits pendant la simulation en fonction du choix de stratégie d'observation, pour produire incrémentalement les éléments de la trajectoire. Par exemple, dans le contexte des modèles de SES, il est possible, en spécifiant une stratégie d'observation planifiée (ou O_{timed}), de suivre l'évolution de la quantité des stocks de ressource d'un espace géographique donné, toutes les semaines, tous les mois ou toutes les années.

6.2.1.3. Méta-modèle du langage L4

Le méta-modèle du langage L4 est représenté par la Figure 72. Il se divise en trois blocs :

- Le bloc « instance » : Etant donné que les structures de données d'un modèle de simulation sont généralement arborescentes. Le bloc « instance » permet à l'utilisateur de spécifier des chemins (ou path, en anglais) pour accéder à n'importe quel morceau d'un modèle.
- Le bloc « type/meta-data » : Ce bloc permet de spécifier comment observer les éléments d'un modèle de simulation, c'est-à-dire les stratégies d'observation. Le choix de cette stratégie dépend du type d'observation de modèle implémenté dans la plateforme de M&S. Si le modèle de simulation implémente le pattern *« observateur »* pour pouvoir récupérer les éléments du modèle, alors l'utilisateur peut spécifier une stratégie évènementielle ($O_{observer}$), si par contre il n'implémente que les accesseurs *« get »*, alors l'utilisateur peut spécifier une stratégie planifiée (O_{timed}).
- Le bloc « observation » : Ce bloc permet de spécifier en fonction de la stratégie d'observation choisie, quand-est-ce qu'on récupère les données correspondant aux parties observables du modèle de simulation ?

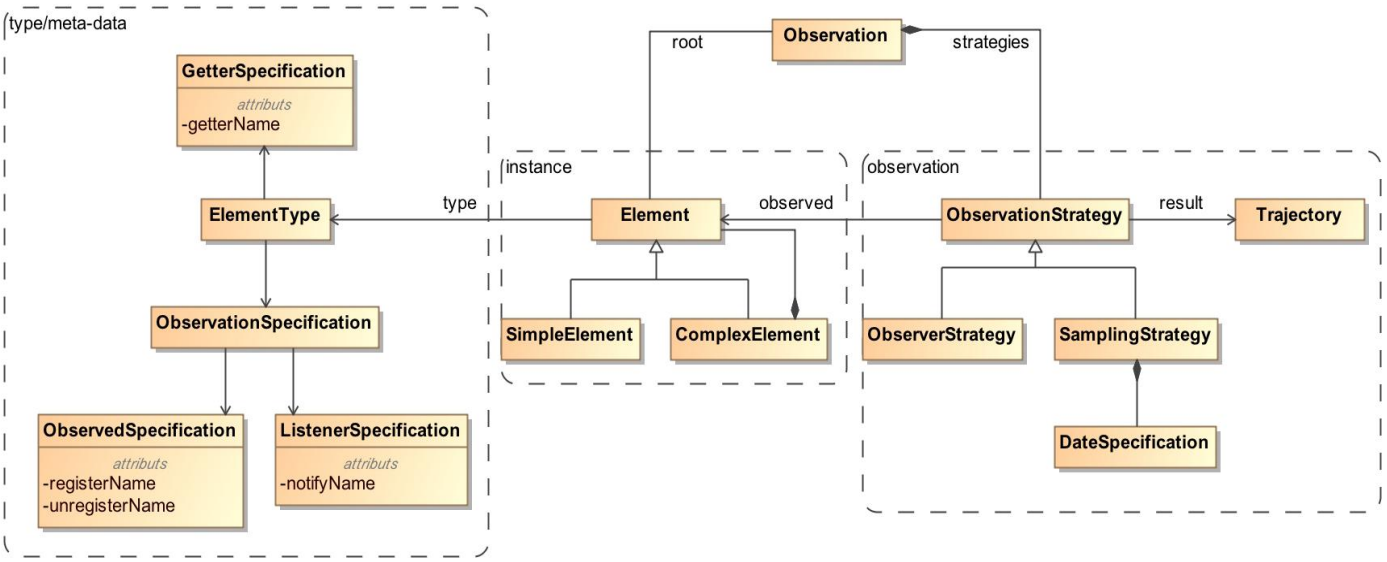


Figure 72. Méta-modèle du langage L4

6.2.1.4. Syntaxe concrète de L4

Comme les DSL d'initialisation de modèle de simulation, nous avons utilisé le langage XText pour définir la syntaxe concrète de L4. La grammaire du langage est présentée en *annexe A.4*. Elle permet à l'utilisateur de:

- Définir le nom de la trajectoire;
- Spécifier le chemin qui mène aux éléments à observer;
- Spécifier le type de stratégie d'observation des éléments à adopter selon le type d'observation disponible dans la plateforme de M&S ;
- Spécifier la stratégie d'observation (la stratégie « observer » ou la stratégie d'échantillonnage) afin de récupérer incrémentalement les éléments à observer.

6.2.1.5. Sémantique comportementale de L4

Le code informatique généré lors de la spécification du modèle d'observation permet d'échantillonner le modèle de simulation selon un choix de stratégie d'observation afin de produire incrémentalement les éléments de la trajectoire du modèle de simulation. Le générateur de code du langage L4 a été spécifié avec le langage XTend. Le code complet est présenté en *annexe B.3.*

6.2.1.6. Illustration du langage L4 sur le modèle ECEC

A l'aide de la syntaxe concrète du langage L4, l'utilisateur peut spécifier, indépendamment de la technologie et de la plateforme de M&S utilisées, les sous-parties du modèle ECEC qu'il voudrait suivre pendant la simulation avec une stratégie d'observation.

Entre autres, le Code 26 spécifie le chemin à partir de la racine de la structure de données du modèle de simulation et permet d'extraire la quantité d'énergie de chaque prédateur tous les 35 pas de temps pendant la simulation.

```
tracking "foragerEnergy"{
    //root
    root: "ecec.system.Root";
    //path
    path: /population:Population/subject:Subject/resource:Resource ;

    //extraction of the elements of the model
    methodTracking population{
        getfor:Root.getPopulations();
    }
    methodTracking subject{
        getfor:population.getSubjects();
    }
    methodTracking resource{
        getfor:subject.getResources();
    }

    return outputForager = {subject resource  subject.getQuantity(resource)}

    //observation strategy
    strategy observing "forager"(outputForager);
    stepTime = 12 month;

    }
```

Code 26. Observation de l'évolution de la quantité d'énergie de chaque prédateur avec L4

Le Code 27 spécifie le chemin à partir de la racine de la structure de données du modèle de simulation, et permet d'extraire, en cas de changement d'état d'une plante, sa quantité de biomasse pendant la simulation.

```
tracking "biomass"{
    //root
    root: "ecec.system.Root";
    //path
    path: /territorySpace:TerritorySpace/place:Place/resource:Resource ;

    //extraction of the elements of the model
    methodTracking territorySpace{
        getfor:Root.getTerritorySpace("Grid");
    }
    methodTracking place{
        cast:(ResourceHolder)resourceholder;
    }
    methodTracking resource{
        getfor:resourceholder.getResources();
    }

    return outputBiomass = {resourceholder resource  resourceholder.getQuantity(resource)}

    //observation strategy
    strategy observing "biomass"(outputBiomass);
    stepTime = 12 month;

    }
```

Code 27. Observation de l'évolution de la quantité de biomasse de chaque plante avec L4

6.2.2. Le Langage L6

Le langage L6 permet de spécifier la structure des indicateurs ainsi que leurs présentations du côté des utilisateurs.

6.2.2.1. Principe et état de l'art

Un indicateur est une structure de données informatique, qui représente l'information dont les thématiciens ont besoin pour comprendre le fonctionnement du modèle de simulation afin de faire une évaluation ou une prise de décision sur le système étudié. Il peut être ensuite, soit sauvegardé sur un support de données (bases de données, fichiers, etc.) pour d'éventuelles utilisations, soit visualisé à partir d'un outil de visualisation interactive (histogramme, courbe, tableau, etc.).

Le fait d'avoir besoin de spécifier des structures de données et des supports de données nous fait référence au méta-modèle du langage L1. Au fait, pour l'initialisation de modèles de simulation, nous avons mis en œuvre un méta-modèle permettant de spécifier des structures de données à partir des sources de données hétérogènes (Figure 61). Ce méta-modèle peut être exploité pour mettre en œuvre celui du langage L6, sauf qu'au lieu de construire les structures de données à

partir des sources de données, elles sont les indicateurs que les thématiciens souhaitent suivre et ensuite les supports de données sont utilisés pour les présenter visuellement pendant la simulation.

Afin d'identifier les concepts nécessaires à la spécification de la présentation visuelle des indicateurs, nous allons présenter quelques travaux qui ont été menés sur la visualisation des données en général.

Le problème de visualisation ne date pas d'aujourd'hui mais depuis une quarantaine d'années. En effet, des solutions de représentation de données, d'informations, ou de connaissances ne cessent d'évoluer jusqu'à aujourd'hui selon les besoins en termes de visualisation. Dans les années 70, des laboratoires en sciences informatiques ont travaillé sur la visualisation des données scientifiques et cela a permis de mettre en œuvre des méthodes, des techniques et des procédés de représentation de données scientifiques d'ordre métrique et statistique. Dans les années 90, des approches croisées multi-expertes, mettant en relation l'ingénierie et la cognition, ont été proposées afin d'améliorer la compréhension des données étudiées. Ces approches ont donné naissance à un domaine de recherche important sur la représentation graphique de l'information appelée « InfoVis» (Information Visualization)[44] qui consiste à transformer des structures de données en des formes visuelles compréhensibles par les parties prenantes du système. Dans ce sens, Stuart et son équipe ont formalisé le processus de visualisation de données [Card 1999] (Figure 73) afin d'amplifier les connaissances en partant des données brutes ou des tables de données.

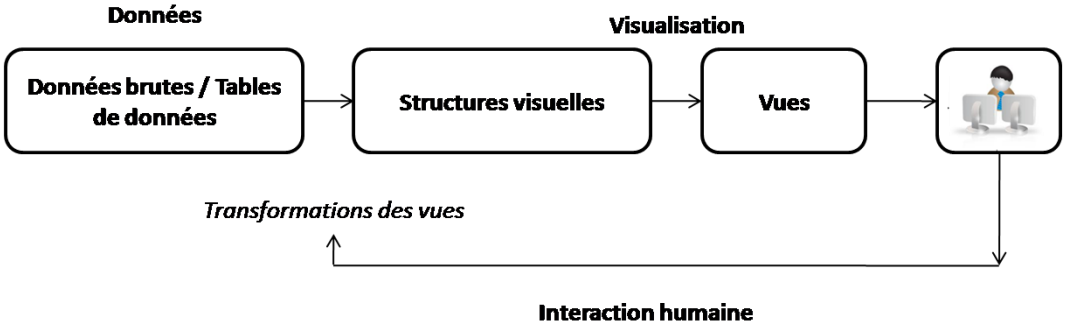


Figure 73. Processus de création d'une visualisation (Figure adaptée de Stuarts K.)

Parallèlement à ces approches, d'autres techniques de visualisation de données ont vu le jour. Entre autres, la technique appelée ''design'' ou ''ROO'' (pour Représentation Orientée Objet), inspirée du concept orienté objet et permet de représenter visuellement les données et l'approche dite par ''points de vues étendus'' [Bihanic et Polacsek 2012] qui a été élaborée pour représenter et maîtriser la visualisation des systèmes d'informations complexes (SIC). Pour ce faire, Bihanic

[44]http://www.infovis-wiki.net/index.php?title=Main_Page

et Polacsek ont utilisé l'IDM et ont défini des DSL composés d'un langage de modélisation conforme à MOF pour représenter les données du SIC (langage source), un langage dénommé VSML (a Visual Semantic Unified Modeling Language) conforme également à MOF pour représenter graphiquement le modèle du SIC (langage cible) et un langage de transformation de modèles permettant de passer d'un modèle SIC à un modèle de représentation visuelle (Figure 74).

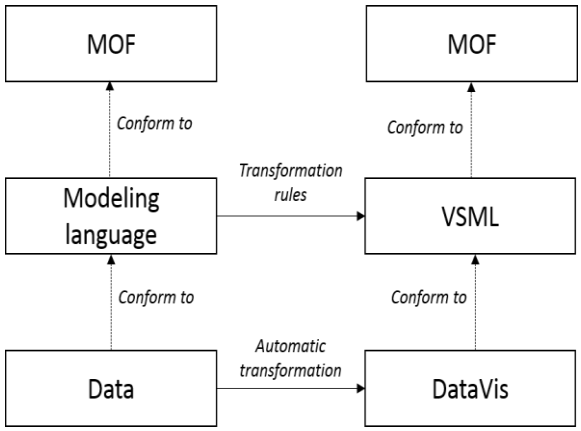


Figure 74. Transformation du langage de modélisation vers le langage VSML

En informatique comme en visualisation scientifique, on est arrivé au besoin de séparer ce qu'on veut visualiser (le modèle) de comment le visualiser (la vue). Pour ce faire, Trygve Reenskaug a élaboré l'architecture Modèle – Vue – Contrôleur (MVC) [Burbeck 1992], qui est un cadre normalisé dont le principe est de séparer les données (le modèle), la présentation (la vue), et le traitement (le contrôleur) afin de faciliter la manipulation ainsi que la visualisation des données volumineuses.

Aussi, nous pouvons nous inspirer de ces techniques de visualisation de données pour élaborer le méta-modèle dédié à la spécification des indicateurs et sa présentation visuelle. Notamment, la technique des points de vues étendus élaboré par Bihanic et Polacsek nous permet d'identifier les concepts utiles à la transformation des indicateurs en des formes graphiques (courbes, graphes, histogrammes, etc.). Le processus de visualisation de données proposé par Stuart et son équipe ainsi que l'architecture ''Modèle – Vue'' (MV) du modèle de conception MVC encouragent à distinguer et à séparer la spécification de la structure des indicateurs (comme modèles) de leur présentation (comme vues) dans le processus d'observation de modèles de simulation.

6.2.2.2. Méta-modèle du langage L6

A partir de ce qui précède, nous proposons un méta-modèle composé de deux parties, à savoir :

- Une partie permettant à l'utilisateur de spécifier la structure des indicateurs ;

- Une autre partie pour spécifier les puits de données (ou Datasink en anglais) en charge de faire la présentation visuelle des indicateurs spécifiés ou son stockage sur un support de données. Les indicateurs peuvent être visualisés ou stockés en même temps dans différents supports de données.

Nous proposons par la Figure 75, la structure globale du méta-modèle du langage L6, tels que les concepts « *ElementType* » et « *ConnectionSpecification* » ont les mêmes structures que celles du méta-modèle du langage L1.

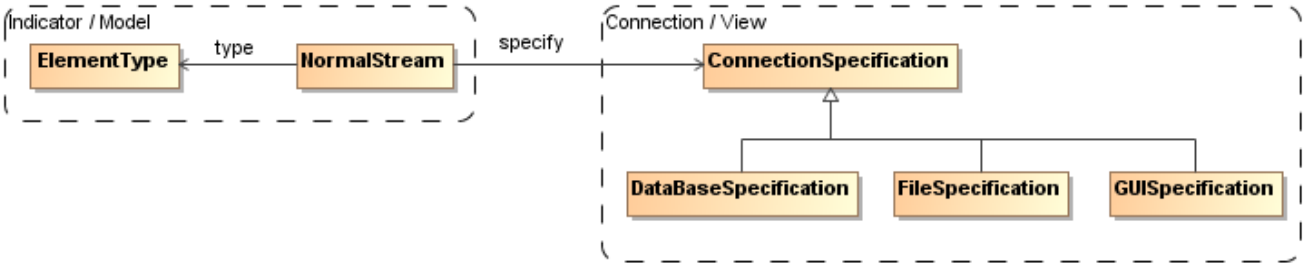


Figure 75. Vision globale du méta-modèle du langage L6

6.2.2.3. Syntaxe concrète de L6

La spécification de la structure des indicateurs ainsi que sa présentation visuelle est possible grâce à la syntaxe concrète que nous avons implémentée avec XText. La grammaire que nous avons élaborée, permet à l'utilisateur de spécifier d'une part les indicateurs, en utilisant le bloc *DataSetSpecification;* d'autre part les supports de données où ils seront présentés visuellement, en utilisant le bloc *ConnectionSpecification*. La grammaire du langage L6 est présentée en *annexe A.5*.

Afin d'illustrer l'utilisation du langage, considérons une structure de données très simple que nous voulons présenter à partir d'un composant graphique « *JTable* » du langage Java. Cette structure de données décrit la surface des habitats en hectare dans un espace géographique à un moment donné. Pour afficher ces données, qui représentent les indicateurs, dans un composant graphique, nous devons spécifier :

- Le type de langage à générer, du code en Java par exemple ;
- La classe de modèle « *DefaultTableModel* » ainsi que les paramètres de son constructeur ou les méthodes associées à celui-ci comme « *addColumn()* », « *addRow()*», *etc.* afin de charger les données dans une instance de modèle de tableau;

- La classe du composant graphique « *JTable* », pour afficher les données dans un tableau ;
- La relation « *setModel* » entre le composant « *JTable*» et le modèle de données « *DefaultTableModel* » pour que les données récupérées soient affichées via le composant « *JTable* ».

La spécification élaborée avec le langage L6 permet de générer automatiquement du code exécutable et permet de visualiser les données dans un tableau (Figure 76).

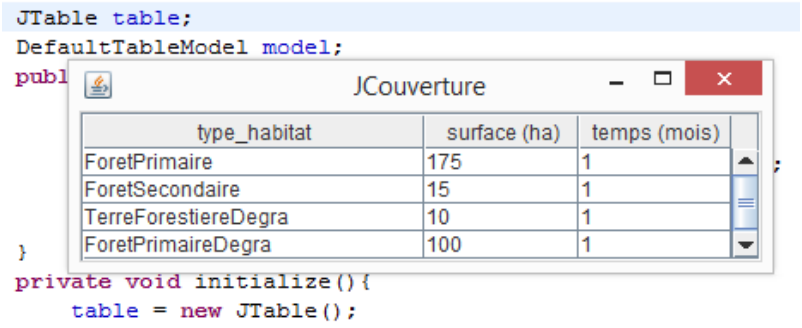
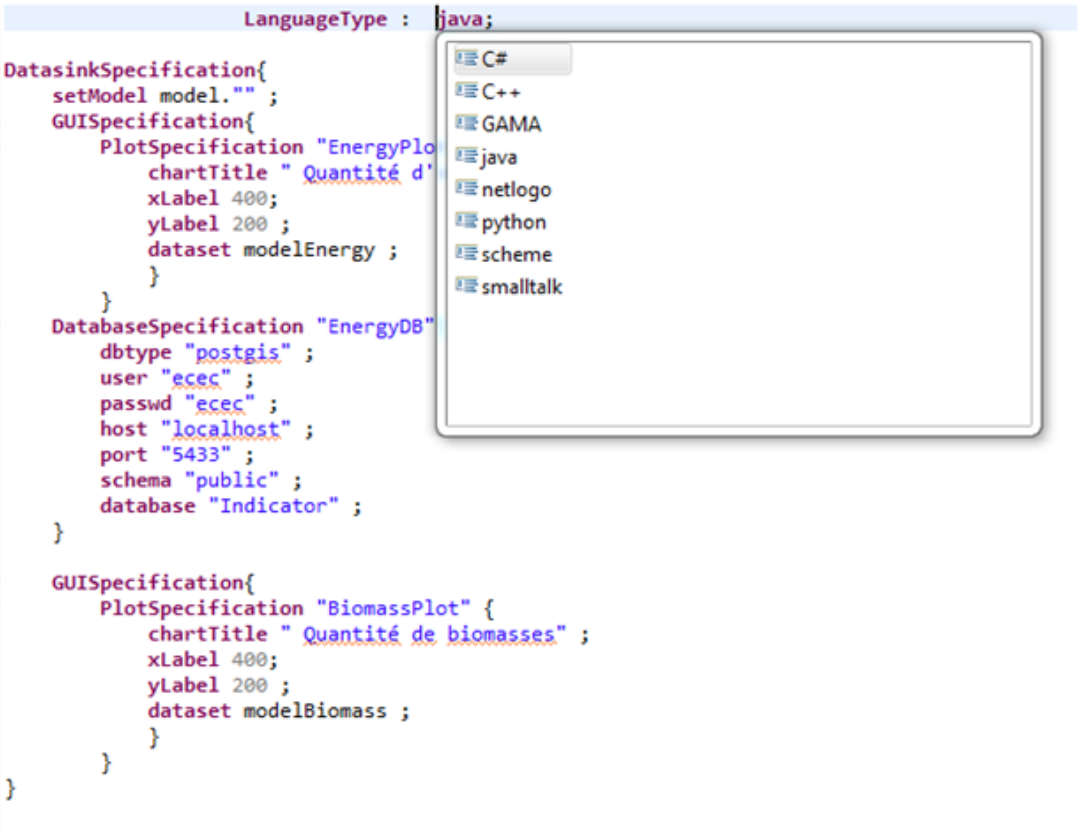


Figure 76. Exemple de visualisation générée à partir de L6

6.2.2.4. Illustration du langage L6 sur le modèle ECEC

A l'aide de la syntaxe concrète du langage L6, l'utilisateur peut spécifier, indépendamment de l'architecture et de la plateforme de M&S utilisées la présentation visuelle des indicateurs qu'il souhaite observer et analyser lors de la simulation. Supposons que nous voulons afficher la quantité d'énergie de chaque prédateur et la quantité de biomasse à l'aide d'un composant graphique en Java et les stocker en même temps dans une base de données. Pour ce faire, il suffit de choisir le type de langage à générer et de spécifier des connections auxquelles les données, notamment les indicateurs seront présentés. Un exemple reflétant cette spécification est montré par le Code 28.



Code 28. Spécification de la présentation de l'évolution de la quantité de biomasse et la quantité d'énergie de chaque prédateur avec L6

6.2.3. Le Langage L5

Le langage L5 permet de transformer les trajectoires issues de la sortie du langage L4, en indicateurs souhaités par les thématiciens (spécifiés à partir du langage L6 – Section 3.3).

6.2.3.1. Principe du langage L5

Les trajectoires issues de la simulation sont obtenues de manière incrémentale en fonction de l'évolution du modèle et de la stratégie d'observation définie à partir du langage L4. Avoir un méta-modèle capable de gérer les flux de données devient alors nécessaire pour obtenir les indicateurs à partir des séquences de données issues de la simulation. Pour cela, nous proposons d'exploiter d'une part, l'architecture ''DataFlow'' (ou flux de données) [Silcet al. 2012], qui est une architecture et une technique très répandue dans le domaine du parallélisme depuis les années 70 pour exécuter les transformations à la fois incrémentale et parallèle ; d'autre part, le système de gestion de flux de données (ou ''DSMS'' pour Data Stream Management System) [Golab et Tamer 2003] qui permet de gérer et d'interroger des données dans un flux de données en continu.

Ainsi, pour alimenter les structures de données des indicateurs, nous proposons le processus décrit par la Figure 77. Le principe consiste à récupérer la trajectoire du modèle de simulation issue du langage L4, puis à en faire un prétraitement dans le but d'extraire la partie temporelle, afin d'adapter le format des données à la manipulation de flux de données. L'étape suivante consiste à exécuter une chaîne de transformations spécifiée par l'utilisateur, sur le flux de données tant que les structures de données à manipuler sont disponibles. Les résultats sont mis à jour à chaque arrivée de nouvelles structures de données. La Figure 77 présente le processus de transformation des trajectoires vers les indicateurs.

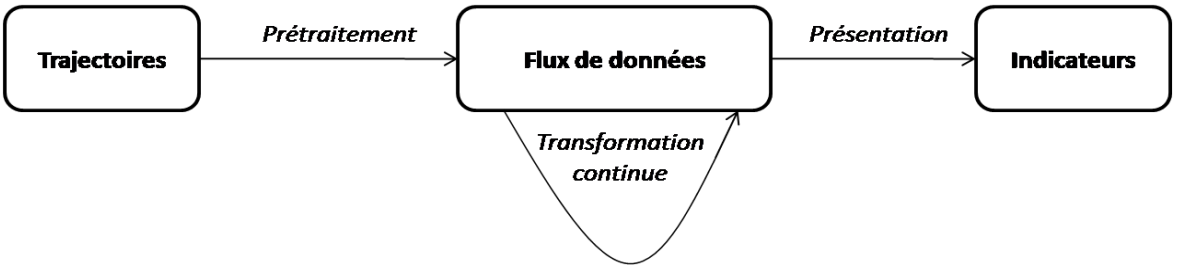


Figure 77. Des trajectoires vers les indicateurs

Deux types de traitements sont donc pris en compte, à savoir:

- Le prétraitement ;
- La manipulation des structures de données pour construire les indicateurs.

a. Prétraitement

Le prétraitement permet d'obtenir des structures de données informatiques à partir des séquences de données indexées par le temps, qui sont récupérées pendant la simulation. Concrètement, pour chaque donnée disponible (simple ou complexe) indexée par le temps, le générateur de code crée automatiquement une structure attributive avec un attribut *temps* et un attribut associé aux données de type simple ou complexe.

b. Manipulation des Structures de Données pour Engendrer les Indicateurs

Manipuler les structures de données issues du prétraitement permet aux thématiciens de construire les indicateurs qu'ils souhaitent observer pendant la simulation. En effet, pour comprendre les corrélations entre les structures de données d'un modèle de simulation ou pour obtenir des indicateurs synthétiques, les thématiciens font généralement du traitement de données et des

analyses statistiques. Ils ont recours à des outils statistiques comme R, STATA, SPSS ou SAS. Ces outils disposent de différents paquets permettant de traiter automatiquement divers calculs, méthodes et modèles statistiques.

Le langage que nous proposons, devra permettre aux thématiciens de faire un travail similaire, sauf que dans le cas d'une simulation de modèle de SES, les structures de données disponibles issues de la transformation des trajectoires ainsi que les indicateurs à construire évoluent par rapport à l'évolution du modèle. Par conséquent, les indicateurs obtenus à partir de diverses opérations (moyenne, médiane, écart type, quantile, etc.) évoluent aussi selon les structures de données disponibles en cours de simulation.

6.2.3.2. Méta-modèle du langage L5

Nous proposons donc, un méta-modèle (Figure 78), inspiré de la gestion de flux de données et qui répond aux attentes des thématiciens, permettant de spécifier des transformations (méthodes statistiques, calculs arithmétiques, etc.) ainsi que les types de données dont elles ont besoin pour fonctionner afin de les exécuter pour obtenir des résultats, chaque fois que les données nécessaires sont disponibles.

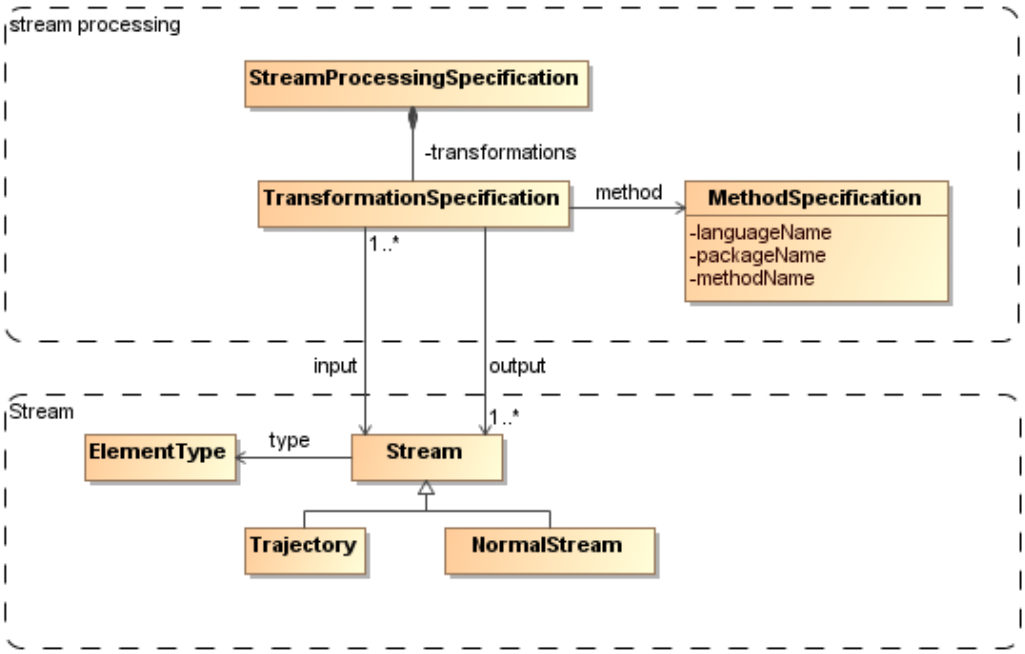


Figure 78. Méta-modèle de L5 pour obtenir les indicateurs

La spécification faite à partir de ce méta-modèle permet d'avoir un graphe dans lequel, les nœuds sont les transformations à effectuer et les arcs servent pour la circulation des flux de données disponibles afin de former les entrées nécessaires à l'activation d'un nœud et leurs sorties (résultats obtenus après l'exécution d'une transformation), qui peuvent être des indicateurs souhaités par les thématiciens. Les sorties d'un nœud peuvent être utilisées comme des entrées d'autres nœuds. Le graphe ainsi obtenu représente la structure de l'exécution de la chaîne de transformations à faire pour générer les indicateurs souhaités.

Par exemple, supposons qu'on veut obtenir l'évolution de la moyenne des surfaces d'un type d'habitat spécifique en hectare dans un espace géographique pendant la simulation. Pour ce faire, il suffit de spécifier une fonction qui permet de calculer la moyenne d'une séquence de données ainsi que le flux de données d'entrée nécessaire pour son exécution. Chaque fois que le modèle de simulation évolue (i.e., de nouvelles données disponibles), la fonction utilise la nouvelle structure de données disponible en entrée et met à jour la moyenne des valeurs observées.

6.2.3.3. Syntaxe concrète de L5

La syntaxe concrète de L5 permet de spécifier une chaîne de transformations permettant de générer du code pour construire les indicateurs souhaités par les thématiciens. Elle a été aussi implémentée avec le langage XText. Du point de vue syntaxique, la grammaire du langage L5 que nous avons présentée en *annexe A.6* est proche du langage L2 (cf. 5.2.3.3) qui est utilisé dans le processus d'initialisation. Il se distingue cependant par sa sémantique. D'une part, le langage L2 permet de générer des structures de données conformes au modèle de simulation à partir des structures de données intermédiaires issues des sources de données hétérogènes. D'autre part, le langage L5 va prendre comme entrées, les données ou le flux de données récupérées à partir du modèle de simulation pendant son exécution pour générer d'autres structures de données pour alimenter les indicateurs que les thématiciens souhaitent visualiser.

6.2.3.4. Sémantique comportementale de L5

Le code généré lors de la spécification de la chaîne de transformation permet selon la technologie utilisée, de mettre en correspondance les trajectoires du modèle issues de L4 et les indicateurs spécifiés à partir de L6, d'instancier ensuite et d'alimenter les structures de données des indicateurs, par les flux de données récupérés. Le générateur de code du langage L5 a été développé à partir du langage XTend. Le générateur de code complet est présenté en *annexe B.4.*

6.2.3.5. Illustration du langage L5 sur le modèle ECEC

Rappelons que le langage L5 permet de décrire les trajectoires nécessaires à la construction des indicateurs souhaités, de décrire les indicateurs à observer et de spécifier comment manipuler ces trajectoires pour obtenir les indicateurs? Divers indicateurs peuvent intéresser les thématiciens lors de la simulation du modèle ECEC, mais nous nous limiterons à présenter quelques exemples afin d'illustrer son utilisation.

- **Evolution de la quantité d'énergie de quelques prédateurs**

Les instructions représentées sur le Code 29 permettent de générer du code en Java pour construire trois indicateurs, à savoir :

- L'évolution de la quantité d'énergie du prédateur dénommé "constrained (1)";
- L'évolution de la quantité d'énergie du prédateur dénommé "constrained (5)";
- L'évolution de la quantité d'énergie du prédateur dénommé "unconstrained (28)".

```
indicator foragerEnergy{
    input{
        int time = Date;
        String subject = trajectory(L4.outputForager.subject);
        String resource = trajectory(L4.outputForager.resource);
        double energy = trajectory(L4. outputForager.subject.getQuantity(resource));
    }
    output{
        report res = L6.modelEnergy("EnergyPlot");
    }
    transformation{
        //filter
        if (subject == "constrained(1)" or subject == "constrained(5)" or subject == "unconstrained(28)" ){
            //computation
            res.xLabel = time;
            res.yLabel = quantity;
        }
    }
}
```

Code 29. Utilisation de L5 pour suivre l'évolution de la quantité d'énergie de quelques prédateurs

L'exécution du code généré à partir du langage L5 (Code 29) construit les indicateurs souhaités. Ces indicateurs sont représentés sur la Figure 79.

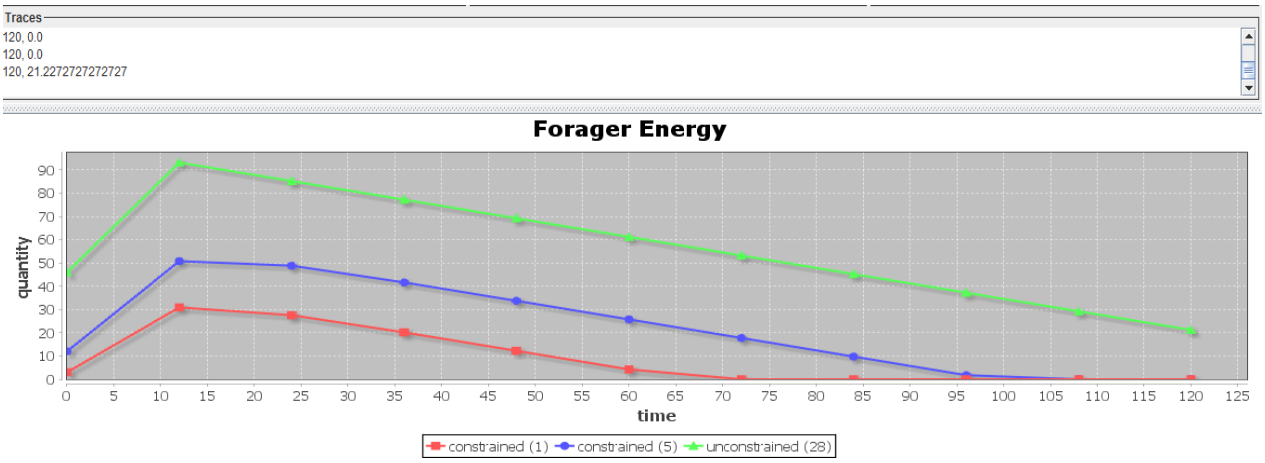


Figure 79. Evolution de la quantité d'énergie de quelques prédateurs

- **Quantité totale de biomasse à un moment donné**

Les instructions représentées sur le code ... permettent de générer du code en Java pour calculer et présenter la quantité totale de biomasse à un instant précis (12 mois).

```
indicator biomassTime{
    input{
        int time = Date ;
        String resourceHolder = trajectory(L4.outputBiomass.resourceHolder);
        String resource = trajectory(L4.outputBiomass.resource);
        double quantity = trajectory(L4.outputBiomass.resourceHolder.getQuantity(resource));
    }
    output{
        report res = L6.modelBiomass("BiomassPlot");
    }
    transformation{
        // filter
         if (time == 12){
            //computation
            res.xLabel = time ;
            res.YLabel = L5.sum(quantity);
         }
    }
}
```

Code 30. Utilisation de L5 pour calculer la quantité totale de biomasse à un instant précis

L'exécution du code généré à partir du langage L5 (Code 30) construit l'indicateur. Ce dernier est représenté sur la Figure 80.

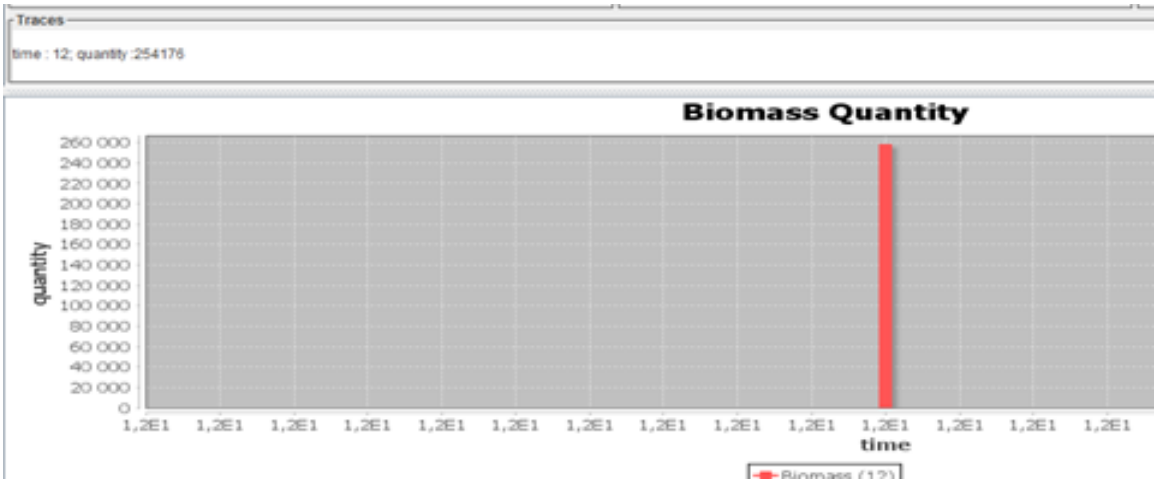


Figure 80. Quantité totale de biomasse à l'instant t = 12 mois

- **Quantité totale de biomasse à deux instants différents de la simulation**

Le principe est toujours le même que celui du précédent, sauf qu'il faudrait ajouter une autre condition (voir la partie "transformation" du Code 31) si on veut observer en même temps la quantité totale respective de biomasse à deux instants différents de la simulation (à l'instant t = 12 mois et t = 24 mois).

```
indicator biomassTimes{
    input{
        int time = Date ;
        String resourceHolder = trajectory(L4.outputBiomass.resourceHolder);
        String resource = trajectory(L4.outputBiomass.resource);
        double quantity = trajectory(L4.outputBiomass.resourceHolder.getQuantity(resource));
    }
    output{
        report res = L6.modelBiomass("BiomassPlot");
    }
    transformation{
        // filter
         if (time == 12 or time == 24){
            //computation
            res.xLabel = time ;
            res.YLabel = L5.sum(quantity);
         }
    }
}
```

Code 31. Utilisation de L5 pour calculer la quantité totale de biomasse à deux instants différents de la simulation

La Figure 81 représente les indicateurs souhaités, à partir de l'exécution du code généré du langage L5 (i.e., le Code 31).

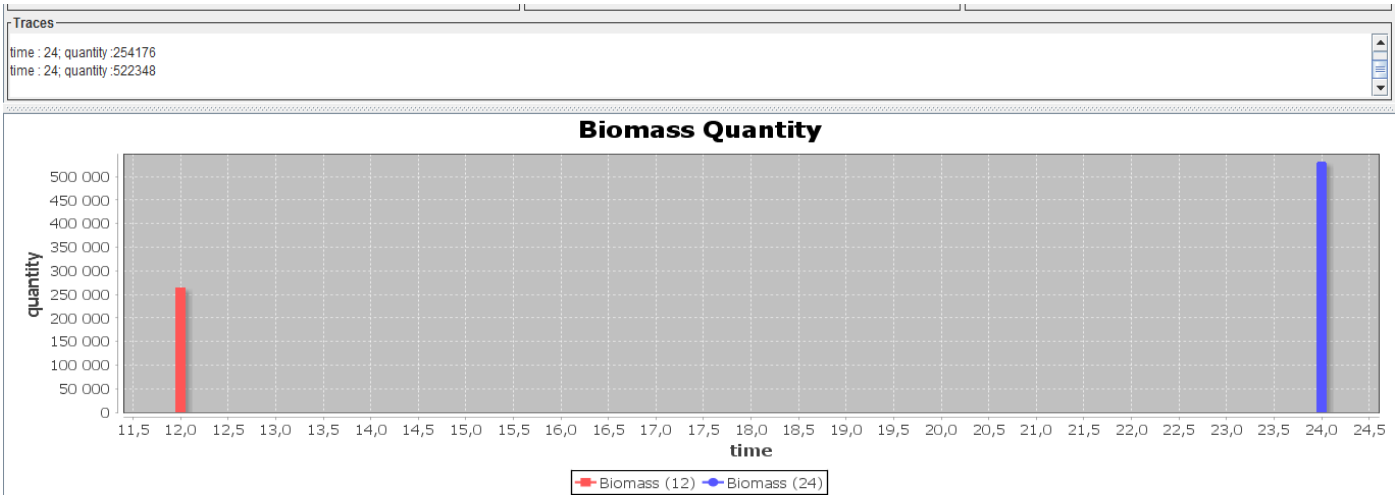


Figure 81. Quantité totale de biomasse au temps t = 12 mois et au temps t = 24 mois

- **Evolution de la quantité de biomasse d'un certain nombre de plantes**

Les instructions représentées sur le Code 32 permettent de générer du code en Java pour présenter l'évolution de la quantité de biomasse sur les coordonnées C1= (30,40), C2= (1,2), C3= (13,15), C4= (30,50) de la grille de taille 50 x 50.

```
indicator biomassCoordinate{
    input{
        int time = Date;
        String resourceHolder = trajectory(L4.outputBiomass.resourceHolder);
        String resource = trajectory (L4.outputBiomass.resource);
        double quantity = trajectory(L4.outputBiomass.resourceHolder.getQuantity(resource));
        }
    output{
        report res = L6.modelBiomass("BiomassPlot");
        }
    transformation{
        //filter
        if (resourceHolder == "(30,50)" or resourceHolder == "(1,2)" or resourceHolder == "(13,15)" or
             resourceHolder == "(30,15)"){
            //computation
            res.xLabel = time;
            res.yLabel = quantity;
        }
    }
}
```

Code 32. Utilisation de L5 pour suivre l'évolution de la quantité de biomasse de quelques endroits

La Figure 82 représente l'évolution de la quantité de biomasse sur quelques endroits de la grille, à partir de l'exécution du code généré du langage L5 (i.e., le Code 32).

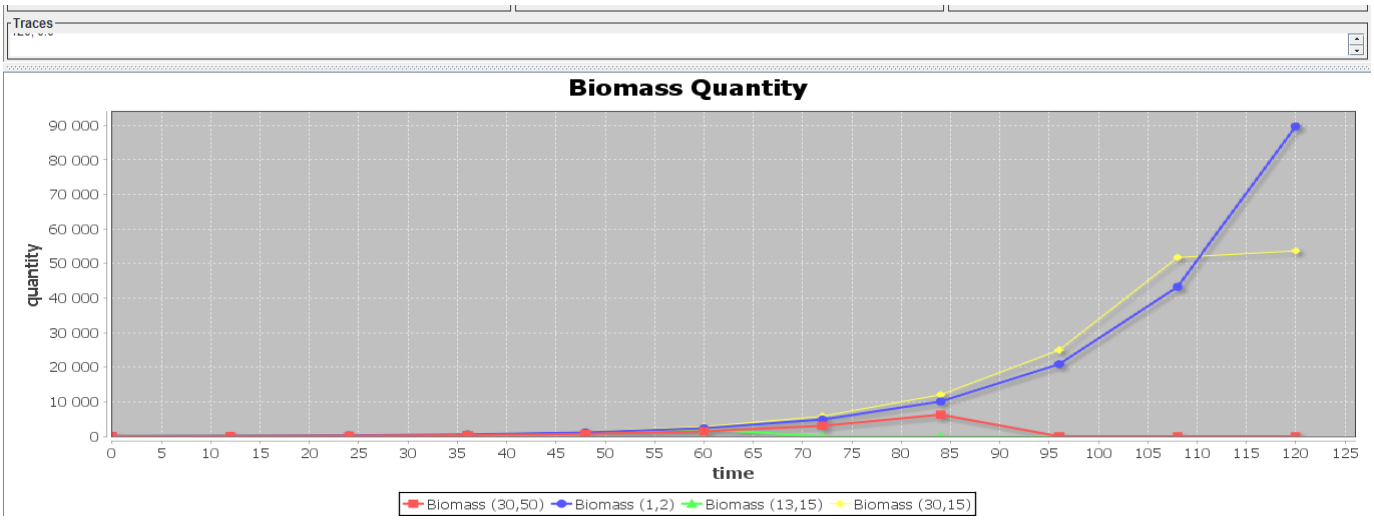


Figure 82. Evolution de la quantité de biomasse sur quelques endroits de la grille

6.3. Conclusion

Dans ce chapitre, nous avons spécifié les langages dédiés (DSL) L4, L5 et L6 en utilisant les concepts d'IDM. Ils permettent de formuler le processus d'observation des modèles de SES de manière générique. Nous avons fourni les concepts nécessaires à l'observation des modèles de SES et spécifié pour chaque méta-modèle, une syntaxe concrète textuelle associée d'une sémantique comportementale afin de pouvoir générer du code exécutable selon la technologie et la plateforme de M&S utilisée. Le code généré permet ensuite de construire et de présenter les indicateurs que les thématiciens souhaitent suivre pendant la simulation.

En effet, pour démontrer la généricité de l'approche, nous l'avons appliquée sur le modèle ECEC afin de répondre aux questions que nous avons posées sur la viabilité du système (cf. 5.3). Avec les langages d'observation que nous avons élaborés (L4, L5 et L6), nous sommes capables d'extraire les sous-parties du modèle de simulation dont on a besoin pour construire les indicateurs avec la possibilité de spécifier des stratégies d'observation. Les éléments observables souhaités du modèle de simulation sont récupérés sous forme de flux (les trajectoires) avec le langage L4 en fonction de la spécification de stratégies d'observation. Il est ensuite possible avec le langage L5 de spécifier des méthodes permettant de les manipuler pour sortir les indicateurs. En utilisant les outils et techniques existants pour la visualisation des données, les indicateurs issus du langage L5 peuvent être visualisés graphiquement ou sauvegardés sur un support de données en les spécifiant avec le langage L6.

Par ailleurs, comme le modèle ECEC est implémenté dans la plateforme de M&S MIMOSA, le prototype que nous avons présenté génère du code en Java. Mais il est tout à fait possible et envisageable de générer d'autres types de code selon la plateforme et le langage utilisés.

Aussi, nous avons pu mettre à disposition des thématiciens, un processus accompagné d'outils permettant de construire et de présenter des indicateurs de manière générique, à partir d'un modèle de simulation.

Chapitre VII. Conclusion générale et perspectives

Le travail de thèse que nous venons de présenter a permis de définir une méthode générique de spécification et de mise en œuvre des scénarios de simulation et des indicateurs que les thématiciens souhaitent suivre. En effet, les modèles de SES sont de plus en plus utilisés pour tester des hypothèses scientifiques, faisant appel, par exemple, à des analyses de sensibilité, ou en appui à la concertation, faisant appel à de l'exploration prospective et participative de scénarii. Cependant, nous avons montré au chapitre II que les modèles de SES sont compliqués et complexes. Pour fonctionner, ils nécessitent d'une part, beaucoup de données et d'informations issues de sources de données hétérogènes. Et d'autre part, les interactions entre les éléments du modèle pendant sa simulation sont imprévisibles. Son exploration devient alors difficile et demande un effort considérable de la part des informaticiens et des modélisateurs.

A chaque fois, l'utilisation des modèles de SES suscite deux questions, à savoir :

- Comment exploiter au mieux les nombreuses données hétérogènes dont disposent les thématiciens pour pouvoir initialiser les modèles de SES ?
- Comment mettre en œuvre les indicateurs relatifs à la fois, aux questions des thématiciens et à la compréhension du fonctionnement du modèle ?

Afin de justifier ce travail de recherche, nous avons présenté au chapitre III, des travaux qui ont été menés par les chercheurs pour traiter les problèmes d'initialisation et d'observation de modèles ou de systèmes en général. Nous avons, par la suite, constaté que malgré les différents efforts, les approches proposées sont relativement ad hoc et aucun cadre générique n'a été encore envisagé : elles dépendent du formalisme utilisé lors de la modélisation et elles sont spécifiques à chaque plateforme de modélisation et de simulation (M&S).

Nous sommes particulièrement partis d'une formulation générique de l'initialisation et de l'observation pour identifier les concepts nécessaires afin d'élaborer les méta-modèles pertinents puis les syntaxes concrètes avec leur opérationnalisation sous la forme de transformations entre données et structures de données. Inspiré des différentes approches proposées dans la littérature, nous avons formulé l'initialisation et l'observation des modèles de simulation en des transformations entre données et structures de données. Cette formulation nous a permis d'utiliser les concepts de l'ingénierie dirigée par les modèles (IDM) que nous avons présentés au chapitre IV afin de mettre à disposition des thématiciens des langages dédiés (DSL) à l'initialisation et l'observation des modèles de SES. Pour montrer la faisabilité, ces DSL ont été implémentés avec

le framework EMF (Eclipse Modeling Framework) avec les outils associés et nous avons illustré le processus complet de notre approche avec le modèle de simulation ECEC. Bien que le prototype que nous avons présenté génère du code en Java, il est tout à fait possible et envisageable de générer d'autres types de code selon la plateforme de M&S et le langage utilisé.

Comme résultat, nous prétendons avoir défini une approche générique de l'initialisation et de l'observation, en fournissant des langages (DSL) permettant de décrire :

- le contenu des sources de données indépendamment des sources ;
- le modèle de simulation indépendamment des plateformes de M&S ;
- comment observer la simulation indépendamment des plateformes de M&S ;
- les indicateurs indépendamment de la forme qu'ils vont prendre pour la présentation.

Les DSL élaborés ont permis de réduire énormément le nombre de lignes de code à écrire pour générer le code informatique nécessaire à l'initialisation et à l'observation d'un modèle.

Par ailleurs, il est important de noter que nous avons considéré deux couches de spécification pour le processus d'initialisation et d'observation d'un modèle de simulation (Figure 83) :

- Une couche abstraite, où les concepts sont complètement génériques car ils sont indépendants des plateformes et des technologies existantes ;
- Une couche spécifique, où la description abstraite devra générer du code, respectivement pour aller chercher les données dans les sources, pour initialiser le modèle, et pour aller présenter les indicateurs dans des supports de données souhaités, selon les plateformes de développement utilisées.

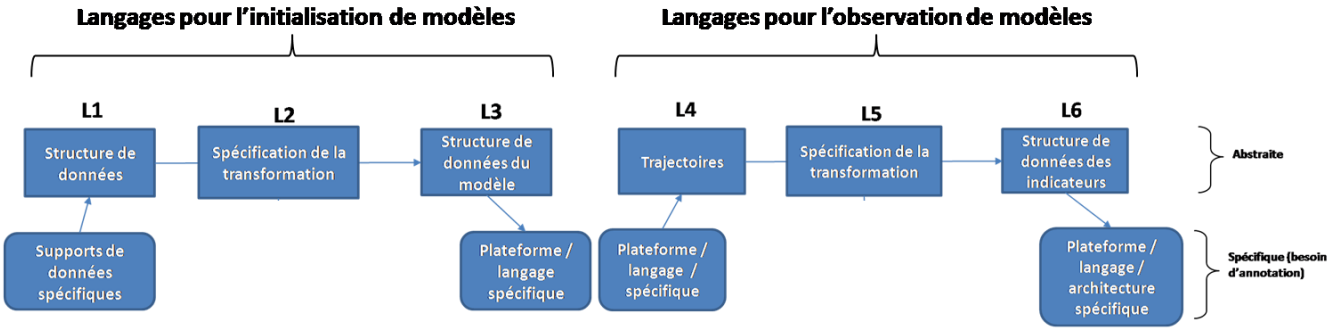


Figure 83. Deux niveaux de spécification dans le processus d'initialisation et d'observation d'un modèle de simulation

Par ailleurs, si on considère un modèle de simulation comme étant un graphe[45], les nœuds du graphe représentent son état à un moment donné de la simulation et les arcs représentent les relations (hiérarchique, voisinage, connexité, etc.) entre les différents éléments. Alors, pendant la simulation, d'une part, l'état de chaque nœud du graphe peut évoluer en fonction du temps et l'ensemble de ces états successifs forme une trajectoire; d'autre part, la structure du graphe est dynamique et évolue éventuellement dans le cas où il y a création de nouveaux éléments ou suppression d'éléments par exemple, au cours de la simulation.

En représentant un modèle de simulation sous la forme d'un graphe :

- L'initialisation revient à construire un graphe initial X_0 dont les nœuds représentent les éléments du modèle de simulation (état initial et paramètres) avec leurs valeurs initiales et les arcs représentent leurs relations. Les méta-modèles génériques des langages L1, L2 et L3 que nous avons élaborés permettent :
    - de spécifier les structures des données nécessaires à l'initialisation du modèle de simulation (L1) ;
    - de décrire sa structure (L3) sous la forme de fonctions $\frac{dx}{dt} = f(x)$;
    - de spécifier comment obtenir la structure du modèle de simulation à partir des structures de données génériques (L2).

  Ces spécifications permettent ensuite de générer le code informatique approprié pour créer les différents éléments du modèle de simulation avec leurs relations et de fournir la valeur initiale de chaque élément à partir des données disponibles. Les structures de ces éléments forment le graphe initial X_0.

- L'observation revient à parcourir le graphe à l'aide des chemins, de récupérer ensuite certains états des éléments du graphe selon une stratégie d'observation ainsi que leur évolution et enfin de les manipuler afin de construire d'autres structures de données qui sont les indicateurs souhaités par les thématiciens. Lorsque la structure du graphe change, il se peut que certains chemins du graphe ne soient plus accessibles, ce qui signifie que l'élément en question n'existe plus dans le modèle de simulation. Les méta-modèles fournis par les langages L4, L5 et L6 permettent :

[45]Graphe : concept mathématique très simple et structure de données informatique très utilisés pour résoudre des problèmes complexes dans de nombreux domaines.

- de spécifier ces besoins ;
- de générer le code informatique permettant de parcourir la structure du modèle de simulation qui représente le graphe ;
- de récupérer incrémentalement l'état de chaque élément qu'on souhaite suivre pendant la simulation selon les stratégies d'observation spécifiées ;
- de construire les structures des indicateurs conformément au langage informatique souhaité ;
- de calculer et de présenter les valeurs associées aux indicateurs, à partir des éléments des trajectoires récupérés des sous-parties observables du MS.

Pour conclure, nous avons pu élaborer une approche générique du processus d'initialisation et d'observation de modèles de simulation complexes, en mettant à disposition des modélisateurs et des thématiciens, les concepts nécessaires indépendamment des plateformes de M&S, les outils et les savoir-faire permettant d'une part, d'exploiter toutes les données hétérogènes issues des thématiciens en vue du paramétrage de divers modèles de simulation et d'autre part, de spécifier et de générer les indicateurs relatifs aux questions des thématiciens à des fins d'aide à la prise de décisions. Cette approche a été validée par les thématiciens et afin de tester sa généricité, elle a été appliquée pour faciliter l'initialisation et l'observation du modèle très complexe Mirana [Aubert et al. 2010] en plus d'ECEC.

Un point que nous avons souligné au début de l'état de l'art (au chapitre III) et qui mériterait une attention particulière à la suite de cette étude, est la question de l'expérimentation des modèles de simulation. En effet, en matière de perspective, il serait intéressant d'avoir des DSL permettant de spécifier que l'on veut lancer un certain nombre de simulations sur des jeux de paramètres et états initiaux dont il faudra décrire la structure, et aussi de spécifier un mécanisme de distribution de la simulation sur des clusters indépendamment de la structure des clusters qu'on a. Pour cela, on pourra partir de l'exemple de SimExplorer [Amblard et al. 2003] ou de son successeur OpenMole [Reuillon et al. 2013].

Un autre point essentiel qui n'a pas été explicité en détail dans cet ouvrage est la question de validation de l'approche et des techniques proposées. Pour ce genre de travail qui propose une généricité d'approche et qui se situe à la frontière des domaines de la modélisation scientifique et de l'ingénierie, le processus de validation nécessite plus de temps et d'interaction avec les

thématiciens. En effet, deux niveaux de validation peuvent être considérés, à savoir : (i) la validation fonctionnelle dont les erreurs (bugs) à corriger se trouvent dans les programmes informatiques (notamment, les compilateurs et les générateurs de code) et (ii) la validation d'usage qui dépend de la validité des programmes informatiques et aussi de la validité des données utilisées et des diverses expérimentations réalisées.

D'autres nouvelles perspectives qui apparaissent à l'issue de cette thèse concernent :

- l'extension de la liste des supports de données possibles en entrée et en sortie, pouvant être pris en compte par les méta-modèles des DSL ;
- la possibilité de spécifier les accès aux différents types de transformation en fonction du langage et de la plateforme de M&S disponible ;
- la possibilité de générer le code d'initialisation et d'observation sur d'autres plateformes de M&S ;
- la formalisation mathématique des sémantiques opérationnelles que nous avons spécifiées ;
- Et enfin, l'amélioration des grammaires (textuelles ou graphiques) fournies dans les DSL selon les demandes des thématiciens.

Références bibliographiques

[Allen 1981] Allen, James F. “An interval-based representation of temporal knowledge.” In *IJCAI*, 81:221–26, 1981.

[Amblard et al. 2003] Amblard, Frédéric, David RC Hill, Stéphan Bernard, Jérome Truffot, and Guillaume Deffuant. “MDA compliant design of SIMEXPLORER a software tool to handle simulation experimental frameworks.” In *Summer Computer Simulation Conference*, 279–84. Society for Computer Simulation International; 1998, 2003.

[Amouroux et al. 2007] Amouroux, Edouard, Thanh-Quang Chu, Alain Boucher, and Alexis Drogoul. “GAMA: an environment for implementing and running spatially explicit multi-agent simulations.” In *Agent Computing and Multi-agent Systems*, 359–71. Springer, 2007.

[Aubert et al. 2010] Aubert, Sigrid, Jean-Pierre Müller, and Julliard Ralihalizara. “MIRANA: a socio-ecological model for assessing sustainability of community-based regulations.” *International Environmental Modelling and Software Society (iEMSs)*, 8, 2010.

[Aubert et Müller 2013] [Aubert et Müller] Aubert, Sigrid and Jean-Pierre Müller. "Incorporating institutions, norms and territories in a generic model to simulate the management of renewable resources", *Artif. Intell. Law, Kluwer Academic Publishers,Hingham, MA, USA*, 47-78, 2013.

[Banos et al. 2005] Banos, Arnaud, Sonia Chardonnel, Christophe Lang, Nicolas Marilleau, and Thomas Thévenin. “Simulating the swarming city: a MAS approach.” In *Proceedings of the 9th International Conference on Computers in Urban Planning and Urban Management, London, June*, 29–30, 2005.

[Banos et al. 2013] Banos, Arnaud, Nicolas Marilleau, MIRO Team, and others. “Improving individual accessibility to the city.” In *Proceedings of the European Conference on Complex Systems 2012*, 989–92. Springer, 2013.

[Barreteau et al.2003] Barreteau, Olivier, M Antona, P d’ Aquino, S Aubert, S Boissau, F Bousquet, Ws Daré, et al. “Our companion modelling approach.” *Journal of Artificial Societies and Social Simulation* 6, no. 2 : 1, 2003.

[Belem et al. 2011a] Belem, Mahamadou, François Bousquet, Jean-Pierre Müller, Didier Bazile, and Harouna Coulibaly. “A participatory modeling method for multi-points of view description of a system from scientist’s perceptions: application in seed systems modeling in Mali and Chile.” *ESSA*, 1–12, 2011.

[Belem et al. 2011b] Belem, Mahamadou, Raphaël J Manlay, Jean-Pierre Müller, and Jean-Luc Chotte. “CaTMAS: A multi-agent model for simulating the dynamics of carbon resources of west african villages.” *Ecological Modelling* 222, no. 20: 3651–61, 2011.

[Bettini 2013] Bettini, Lorenzo. “Implementing domain-specific languages with Xtext and Xtend.” *Packt Publishing Ltd*, 2013.

[Beuret 2010] Beuret Jean-Eudes. “Madagascar : vers une gestion locale des ressources forestières. ” *La médiation au cœur de projets de coopération, Madagascar*, 2010. http://institutionnel.redev.info/outils/dossiers/fiches_mediation/10_MGLOGEN.pdf.

[Bézivin 2003] Bézivin, Jean. “La transformation de modèles.” *INRIA-ATLAS & Université de Nantes*, 13, 2003.

[Bihanic et Polacsek 2012] Bihanic, David, and Thomas Polacsek. “Models for visualisation of complex information systems.” In *16th International Conference on Information Visualisation, Montpellier, France, July 11-13, 2012*, 130–35, 2012.

[Bommel et al. 2012] Bommel, Pierre, Nicolas Becu, Christophe Le Page, and François Bousquet. “Cormas, an agent-based simulation platform for coupling human decisions with computerized dynamics,” 2012.

[Bray et al. 1998] Bray, Tim, Jean Paoli, C Michael Sperberg-McQueen, Eve Maler, and François Yergeau. “eXtensible Markup Language (XML).” *World Wide Web Consortium Recommendation REC-xml-19980210*, 1998.

[Bruel et al. 2015] Bruel, Jean-Michel, Benoit Combemale, Ileana Ober, and Hélène Raynal. “MDE in Practice for Computational Science. ” *International conference on computational science*, June 2015, Reykjavík, Iceland, 2015.

[Burbeck 1992] Burbeck, Steve. “Applications programming in Smalltalk-80 (tm): How to use Model-View-Controller (mvc).” *Smalltalk-80 V2* 5, 1992.

[Card et al. 1999] Card, Stuart K., Jock D. Mackinlay, and Ben Shneiderman, eds. “Readings in information visualization: using vision to think. ” *San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.*, 1999.

[Chow et al. 1994] Chow, Alex Chung Hen, and Bernard P. Zeigler. “Parallel DEVS: A parallel, hierarchical, modular, modeling formalism.” In *Proceedings of the 26th Conference on Winter Simulation*, 716–22. WSC ’94. San Diego, CA, USA: Society for Computer Simulation International, 1994. http://dl.acm.org/citation.cfm?id=193201.194336.

[Chrisment et al. 2005] Chrisment, Claude, Geneviève Pujolle, Franck Ravat, Olivier Teste, and Gilles Zurfluh. “Entrepôts de données.” *Techniques de L’ingénieur. Informatique*, no. H3870, 2005.

[Chu et Allstot 2005] Chu, Min, and David J Allstot. “Elitist nondominated sorting genetic algorithm based RF IC optimizer.” *Circuits and Systems I: Regular Papers, IEEE Transactions On* 52, no. 3: 535–45, 2005.

[Clavel et al. 2008] Clavel, Lucie, Raphaèle Ducrot, and Suzana Sendacz. “Gaming with eutrophication: contribution to integrating water quantity and quality management at catchment level.” *IWRA, 13ème Congrès Mondial de L’eau, Montpellier*, 1–4, 2008.

[Combemale 2008] Combemale, Benoıt. “Ingénierie Dirigée par les Modèles (IDM) État de L’art.” *Management*, 1–19, 2008.

[Combemale et al. 2006] Combemale, Benoit, Sylvain Rougemaille, Xavier Crégut, Frédéric Migeon, Marc Pantel, Christine Maurel, and Bernard Coulette. “Towards rigorous metamodeling.” *MDEIS* 6: 23–27, 2006.

[ComMod 2005] ComMod, Collectif. “La modélisation comme outil d’accompagnement.” *Natures Sciences Sociétés* 13, no. 2: 165–68, 2005. http://www.commod.org.

[Conruyt et al. 2009] Conruyt, Noël, Didier Sébastien, Rémy Courdier, Daniel David, Nicolas Sébastien, and Tiana Ralambondrainy. “Designing an information system for the preservation of the insular tropical environment of Reunion island.” In *Advanced Agent-Based Environmental Management Systems*, 61–90. Springer, 2009.

[Daré et al. 2007] Daré, William’s, Christine Fourage, and Ibrahima Diop Gaye. “positionnement des sociologues dans la démarche de modélisation DOMINO.” *Nouvelles Perspectives En Sciences Sociales* 2, no. 2: 103–26, 2007.

[David et al. 2007] David, Daniel, Denis Payet, Aurélie Botta, Gilles Lajoie, Sébastien Manglou, and Rémy Courdier. “Un couplage de dynamiques comportementales: le modèle DS pour l’aménagement du territoire.” In *JFSMA*, 129–38, 2007.

[David et al. 2009] David, Daniel, Denis Payet, Rémy Courdier, and Yassine Gangat. “XELOC: Un support générique pour la configuration et l’initialisation de systèmes multi-agents.” In *JFSMA*, 251–56, 2009.

[Davidson 2001] Davidson, Donald. “Essays on actions and events: philosophical essays. ” *Vol. 1*. *Oxford University Press*, 2001.

[Drey et al. 2009] Drey, Zoé, Cyril Faucher, Franck Fleurey, Vincent Mahé, and Didier Vojtisek. "Kermeta language reference manual." *Manuscript Available Online Http://www. Kermeta. Org*, 2009.

[El Hamlaoui 2012] El Hamlaoui, Mahmoud. "Etat de l'art sur la gestion de la cohérence de modèles hétérogènes," 2012. http://www.irit.fr/publis/MACAO/RapportIRIT-MEH-et-al.pdf.

[Etienne et Le Page 2002] Etienne, M, and C Le Page. "Modéliser les dynamiques paysagères pour accompagner un projet d'aménagement du territoire: Le cas du Causse Méjean." In *En: Colloque Gérer Les Paysages de Montagne Pour Un Développement Concerté et Durable», Florac*, 5–6, 2002.

[Étienne 2010] Étienne, Michel. "La modélisation d'accompagnement: Une démarche participative en appui au développement durable. " Editions Quae, 2010.

[Farolfi et al. 2008] Farolfi, S, H Gumede, K Rowntree, and N Jones. "Local water governance in South Africa: to which extent participatory approaches facilitate multi-stakeholder negotiations? the Kat river valley experience." In *Proceedings of the XIII World Water Congress, Montpellier*, 1–4, 2008.

[Farolfi et al. 2010] Farolfi, Stefano, Jean-Pierre Müller, and Bruno Bonté. "An iterative construction of multi-agent models to represent water supply and demand dynamics at the catchment level." *Environmental Modelling & Software* 25, no. 10: 1130–48, 2010.

[Gamma et al. 1995] Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. "Design patterns: Elements of reusable object-oriented software. " Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[Gangat et al. 2009] Gangat, Yassine, Daniel David, Denis Payet, and Rémy Courdier. "Modéliser et simuler l'aménagement énergétique d'un territoire par les systèmes multi-agents." In *Convergence Des Réseaux, de l'Informatique, et Du Multimédia Pour Les E-Services (CRIMES)*, 15, 2009.

[Gaudieux et al. 2014a] Gaudieux, Aurélie, Yassine Gangat, Joël Kwan, and Rémy Courdier. "Study of the interactions between stakeholders by a multi-agents system: application to the governance of natural resources in Miarinarivo district (Madagascar)." In *MAS'14 (International Conference on Modeling and Applied Simulation)*, 2014.

[Gaudieux et al. 2014b] Gaudieux, Aurelie, Joel Kwan, Yassine Gangat, and Rémy Courdier. "MASC: Map Sectors Creator a tool to help at the configuration of multi-agents systems for everyone." In *Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH), 2014 International Conference On*, 836–44. IEEE, 2014.

[Gayral et Grandemange 1992] Gayral, Françoise, and Philippe Grandemange. "Une ontologie du temps pour le langage naturel." In *Proceedings of the 14th Conference on Computational linguistics-Volume 1*, 295–302. Association for Computational Linguistics, 1992.

[Golab et Özsu 2003] Golab, Lukasz, and M Tamer Özsu. "Issues in data stream management." *ACM Sigmod Record* 32, no. 2: 5–14, 2003.

[Golshani et al. 1998] Golshani, Forouzan, Oris D Friesen, and Thomas H Howell. "Method for converting a database schema in relational form to a schema in object-oriented form. " Google Patents, 1998.

[Goupy 2001] GOUPY Jacques. "Introduction aux plans d'expériences." Dunod, Paris, 2001.

[Graham 2009] Graham, Shawn. "Behaviour Space: Simulating roman social life and civil violence." *Digital Studies/Le Champ Numérique* 1, no. 2, 2009.

[Greenfield et Short 2003] Greenfield, Jack, and Keith Short. "Software factories: Assembling applications with patterns, models, frameworks and tools." In *Companion of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages,*

and Applications, 16–27. ACM, 2003.

[Gurung et al. 2006] Gurung, Tayan Raj, Francois Bousquet, and Guy Trébuil. "Companion modeling, conflict resolution, and institution building: Sharing irrigation water in the Lingmuteychu Watershed, Bhutan." *Ecology and Society* 11, no. 2: 36, 2006.

[Halpin et Proper 1995] Halpin, Terry A, and Henderik Alex Proper. "Database schema transformation and optimization." In *OOER'95: Object-Oriented and Entity-Relationship Modeling*, 191–203. Springer, 1995.

[Hervé et Rivière 2015] Hervé, Dominique, and Mylène Rivière. "L'interdisciplinarité s' invite dans les systèmes complexes: Les journées de Rochebrune." *Natures Sciences Sociétés* 23, no. 1: 54–60, 2015.

[Hill et Vickers 2005] Hill, Gillian, and Steven Vickers. "A language for configuring multi-level specifications." *Theoretical Computer Science* 351, no. 2: 146 – 166, 2006.

[Jézéquel et al. 2012] Jézéquel, Jean-Marc, Benoit Combemale, and Didier Vojtisek. "Ingénierie Dirigée par les Modèles : Des concepts à la pratique... " *Edited by Ellipses. Références Sciences. Ellipses*, 2012.

[Jouault et Bézivin 2006] Jouault, Frédéric, and Jean Bézivin. "KM3: a DSL for metamodel specification." In *Formal Methods for Open Object-Based Distributed Systems*, 171–85. Springer, 2006.

[Kamp 1981] Kamp, Hans. "Evénements, représentations discursives et référence temporelle." *Langages*, no. 64: 39–64, 1981.

[Kayser 1990] Kayser, Daniel. "Thruth and the interpretation of natural language: a non-monotonic variable-depth approach." In *ECAI*, 392–97, 1990.

[Le Page et al. 2012] Le Page, Christophe, Nicolas Becu, Pierre Bommel, and François Bousquet. "Participatory agent-based simulation for renewable resource management: The role of the Cormas simulation platform to nurture a community of practice." *Journal of Artificial Societies and Social Simulation* 15, no. 1: 10, 2012.

[Lima et al. 2013] Lima, Tiago, Tiago Carneiro, Sergio Faria, PM Silva, and Miguel Pessoa. "TerraME GIMS: An Eclipse plug-in for environmental modeling." In *Developing Tools as Plug-ins (TOPI), 2013 3rd International Workshop On*, 37–42. IEEE, 2013.

[Mathevet et al. 2003] Mathevet, Raphaël, François Bousquet, Christophe Le Page, and Martine Antona. "Agent-based simulations of interactions between duck population, farming decisions and leasing of hunting rights in the Camargue (Southern France)." *Ecological Modelling* 165, no. 2: 107–26, 2003.

[McDermott 1982] McDermott, Drew. "A temporal logic for reasoning about processes and plans." *Cognitive Science* 6, no. 2: 101–55, 1982.

[Minsky 1965] Minsky, Marvin. "Models, minds, machines." In *Proceedings of the IFIP Congress*, 45–49, 1965.

[MOF 2006] MOF. "Meta Object Facility (MOF) 2.0 Core Specification." OMG, 2001. http://www.omg.org/cgi-bin/doc?formal/2006-01-01.

[Mohamed 2014] Mohamed Youssfi. "Design patterns, Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)." *Maroc, Support de cours*, 2014. http://fr.slideshare.net/mohamedyoussfi9/cours-design-pattern-m-youssfi-partie-2-observer

[Molnár et al. 2007] Molnár, Zoltán, Daniel Balasubramanian, and A Lédeczi. "An introduction to the generic modeling environment." In *Proceedings of the TOOLS Europe 2007 Workshop on Model-Driven Development Tool Implementers Forum*, 2007.

[Monteil et al. 2008] Monteil, C, C Simon, S Ladet, D Sheeren, M Etienne, and A Gibon. "Participatory modelling of social and ecological dynamics in mountain landscapes subjected to spontaneous ash reforestation." In *Modelling Environmental Dynamics*, 199–222. Springer, 2008.

[Müller 2007] Müller, Jean-Pierre. "Mimosa: Using ontologies for modelling and simulation." In *GI Jahrestagung (1)*, 227–31, 2007.

[Müller 2004] Müller, Jean-Pierre. "The Mimosa generic modelling and simulation platform: The case of multi-agent systems." In *5th Workshop on Agent-Based Simulation*, 77–86, 2004.

[Müller et al. 2011] Müller, Jean-Pierre, Hasina Lalaina Rakotonirainy, and Dominique Hervé. "Towards a description logic for scientific modeling." In *KEOD*, 183–88, 2011.

[Naivinit et al. 2010] Naivinit, W, Christophe Le Page, Guy Trébuil, and Nantana Gajaseni. "Participatory agent-based modeling and simulation of rice production and labor migrations in northeast Thailand." *Environmental Modelling & Software* 25, no. 11: 1345–58, 2010.

[Napoleone et al. 2008] Napoleone, Martine, Jacques Lasseur, and Michel Etienne. "Devices based on models to accompany stakeholders in enhancing collective learning and action on livestock farming systems." *IFSA 2008*, 6–10, 2008.

[OMG 2014] OMG. "About the Object Management Group." 2014. http://www.omg.org.

[Ostrom 2009] Ostrom, Elinor. "Understanding institutional diversity. " *Princeton university press*, 2009.

[Pan et Hobbs 2005] Pan, Feng, and Jerry R Hobbs. "Temporal aggregates in OWL-Time." In *FLAIRS Conference*, 5:560–65, 2005.

[Penot 2012] Penot, Éric. "Exploitations agricoles, stratégies paysannes et politiques publiques: Les apports du modèle Olympe. " *Editions Quae*, 2012.

[Pepper et Smuts 2002] Pepper, John W, and Barbara B Smuts. "A mechanism for the evolution of altruism among Nonkin: Positive assortment through environmental feedback." *The American Naturalist* 160, no. 2: 205–13, 2002.

[Polhill et al. 2014] Polhill, Gary, Peter Edwards, Terry Dawson, Dawn Parker, Michael Barton, Alexey Voinov, Tatiana Filatova, Edoardo Pignotti, Kirsten Robinson, and Xiongbing Jin. "Towards metadata standards for sharing simulation outputs." In *Social Simulation Conference*, 2014.

[Premkumar et Potter 1995] Premkumar, G, and Michael Potter. "Adoption of Computer Aided Software Engineering (CASE) technology: An innovation adoption perspective." *ACM SIGMIS Database* 26, no. 2–3: 105–24, 1995.

[Quesnel et al. 2012] Quesnel, Gauthier, Ronan Trepos, and Éric Ramat. "Observations of Discrete Event Models." In *SIMULTECH*, edited by Nuno Pina, Janusz Kacprzyk, and Mohammad S. Obaidat, 32–41. SciTePress, 2012.

[Rakotonirainy et al. 2014] Rakotonirainy, Hasina Lalaina, Jean-Pierre Müller, and Bertin Olivier Ramamonjisoa. "Towards a generic framework for the initialization and the observation of socio-environmental models." In *Model and Data Engineering - 4th International Conference, MEDI 2014, Larnaca, Cyprus, September 24-26, 2014*, 45–52, 2014.

[Rakotonirainy et al. 2015] Rakotonirainy, Hasina Lalaina, Jean-Pierre Müller, and Bertin Olivier Ramamonjisoa. "A generic approach for initializing complex models." *The AUST International Conference on Technology, Abuja, Nigeria, October 12-13,* 2015.

[Rakotonirainy et al. 2015] Rakotonirainy, Hasina Lalaina, Jean-Pierre Müller, and Bertin Olivier Ramamonjisoa. “Model Driven Engineering, applied to observation problems of socio-environmental models.” *The AUST International Conference on Technology, Abuja, Nigeria, October 12-13*, 2015.

[Rakotonirainy et al. 2015] Rakotonirainy, Hasina Lalaina, Jean-Pierre Müller, and Bertin Olivier Ramamonjisoa. “Ingénierie Dirigée par les Modèles (IDM) appliquée à l’observation des modèles socio-environnementaux.” *4ème Conférence en IngénieriE du logiciel, Bordeaux, France, June 9-11*, 2015.

[Ramamonjisoa et al. 2012] Ramamonjisoa, Bruno, Hervé Rakoto Ramiarantsoa, and Thorkil Casse. “La loi Gelose et le transfert de gestion des ressources naturelles à Madagascar.” *Les Cahiers d’Outre-Mer*, 257 : 5-10, 2012.

[Redman et al. 2004] Redman, C.L., J.M. Grove, and L. H. Kuby. “Integrating social science into the Long-Term Ecological Research (LTER) Network: Social dimensions of ecological change, ecological dimensions of social change.” *Ecosystems* 7: 161–71, 2004.

[Reuillon et al. 2013] Reuillon, Romain, Mathieu Leclaire, and Sebastien Rey-Coyrehourcq. “OpenMOLE, a workflow engine specifically tailored for the distributed exploration of simulation models.” *Future Generation Computer Systems* 29, no. 8: 1981–90, 2013.

[Silc et al. 2012] Silc, Jurij, Borut Robic, and Theo Ungerer. “Processor architecture: From dataflow to superscalar and beyond.” *Springer Science & Business Media*, 2012.

[Simon et Etienne 2010] Simon, C, and M Etienne. “A companion modelling approach applied to forest management planning.” *Environmental Modelling & Software* 25, no. 11: 1371–84, 2010.

[Smith et al. 1995] Smith, Laron, Richard Beckman, and Keith Baggerly. “TRANSIMS: Transportation Analysis and Simulation System.” Los Alamos National Lab., NM (United States), 1995.

[Soley et al. 2000] Soley, Richard, and others. “Model Driven Architecture.” *OMG White Paper* 308, no. 308: 5, 2000.

[Speed 2005] Speed, Terry P. “Iterative Proportional Fitting.” *Encyclopedia of Biostatistics*, 2005.

[Steinberg et al. 2008] Steinberg, Dave, Frank Budinsky, Ed Merks, and Marcelo Paternostro. “EMF: Eclipse Modeling Framework. ” Pearson Education, 2008.

[Taillandier 2013] Taillandier, Patrick. “GAMAGraM: Modélisation Graphique Sous GAMA.” In *Masyco 2013*, 2013.

[Therond et al. 2014] Therond, Olivier, Christophe Sibertin-Blanc, Romain Lardy, Benoit Gaudou, Maud Balestrat, Yi Hong, Thomas Louail, et al. “Integrated modelling of social-ecological systems: The MAELIA high-resolution multi-agent platform to deal with water scarcity problems.” In *Proceedings of the 7th International Congress on Environmental Modelling and Software, June*, 15–19, 2014.

[Tisue et Wilensky 2004] Tisue, Seth, and Uri Wilensky. “Netlogo: A simple environment for modeling complexity.” In *International Conference on Complex Systems*, Vol. 21. Boston, MA, 2004.

[Turki 2008] Turki, Skander. “Ingénierie système guidée par les modèles: application du standard IEEE 15288, de l’architecture MDA et du langage SysML à la conception des systèmes mécatroniques.” Université du Sud Toulon Var, 2008.

[UNICEF 2010] UNICEF. " Le développement humain durable." 2010. https://www.unicef.fr/sites/default/files/userfiles/dev-humain-durable-11.pdf

[Vangheluwe 2000] Vangheluwe, H. “DEVS as a common denominator for hybrid systems modelling.” In IEEE International Symposium on Computer-Aided Control System Design, 129–34. IEEE Computer Society Press, Anchorage, Alaska, n.d., 2000.

[Vega et al. 2006] Vega, Daniel, Ronald Peñarrieta, and Nicolás Faysse. “Proceso multi-actor y uso de juego de roles: concertación para manejar impactos de la urbanización sobre infraestructura de Riego En Tiquipaya, Cochabamba-Bolivia.” *Experiencias Del Proyecto Negowat En Bolivia: Facilitando Negociaciones Sobre El Acceso Al Agua y Uso de La Tierra En Zonas Periurbanas Cochabamba: Centro Agua-UMSS*, 1: 161–98, 2006.

[Völter et al. 2013] Völter, M, S Benz, C Dietrich, B Engelmann, M Helander, LC Kats, and G Wachsmuth. “DSL engineering: Designing, implementing and using domain-specific languages. Dslbooks. Org,” 2013.

[Warmer et Kleppe 1998] Warmer, Jos B, and Anneke G Kleppe. “The Object Constraint Language: Precise modeling with UML (Addison-Wesley Object Technology Series),” 1998.

[Widom 1995] Widom, Jennifer. “Research problems in data warehousing.” In *Proceedings of the Fourth International Conference on Information and Knowledge Management*, 25–30. ACM, 1995.

[Wiederhold 1992] Wiederhold, Gio. “Mediators in the architecture of future information systems.” *Computer* 25, no. 3: 38–49, 1992.

[Winskel 1993] Winskel, Glynn. “The formal semantics of programming languages: An introduction. ” *MIT press*, 1993.

[Worrapimphong 2005] Worrapimphong, K. “Companion modelling for razor clam solen regularis conservation at Don Hoi Lord.” *Samut Songkhram Province, Chulalongkorn University, Bangkok, Thaïlande*, 2005.

[Worrapimphong et al. 2010] Worrapimphong, Kobchai, Nantana Gajaseni, Christophe Le Page, and François Bousquet. “A companion modeling approach applied to fishery management.” *Environmental Modelling & Software* 25, no. 11: 1334–44, 2010.

[Yesson et al. 2007] Yesson, Chris, Peter W Brewer, Tim Sutton, Neil Caithness, Jaspreet S Pahwa, Mikhaila Burgess, W Alec Gray, et al. “How global is the global biodiversity information facility?” *PLoS One* 2, no. 11: e1124, 2007.

[Zeigler et al. 2000] Zeigler, Bernard P, Herbert Praehofer, and Tag Gon Kim. “Theory of modeling and simulation: Integrating discrete event and continuous complex dynamic systems.” *Academic press*, 2000.

Glossaire

Carte pivot : Une combinaison de cartes raster et vectorielles.

Carte raster : Une représentation géographique obtenue en numérisant et en géo-référençant une carte papier thématique traditionnelle.

Carte vectorielle : Une représentation géographique dont les données sont numérisées et traitées dans un format permettant l'ajout d'informations complémentaires, non comprises dans une carte de type raster.

Expérimentation : Une succession de simulations avec, en général, des paramètres initiaux différents pour chaque simulation.

Indicateurs : Un indicateur est une information construite à partir d'un observable pour pouvoir mesurer l'évolution de différents éléments qui intéressent les thématiciens pour répondre aux questions qu'ils se posent dans le SES à des fins d'évaluation et/ou d'aide à la prise de décision.

Initialisation de modèle : Pour pouvoir construire la fonction du temps $f(t)$, il est nécessaire de simuler le modèle à partir d'un temps initial t_0 pour lequel on connait son état. Nous considérons qu'initialiser un modèle de simulation revient ainsi à spécifier l'état x_0 du modèle à l'instant initial t_0, mais aussi à spécifier la valeur des différents paramètres de la fonction f.

Modèle complexe : Un modèle dont les interactions entre ses éléments sont non-linéaires. Par conséquent, son comportement global est non réductible à la composition des comportements individuels de ces éléments.

Modèle compliqué : Un modèle qui considère de nombreux éléments pour fonctionner.

Modèle de simulation : Dans le cadre de la théorie de la modélisation et de la simulation (TMS), Bernard Zeigler a défini le modèle de simulation comme étant une représentation simplifiée d'un système réel basée sur la définition d'un ensemble d'instructions, de règles, d'équations ou de contraintes permettant de générer un comportement lors de la simulation [Zeigler 1976]. De manière générale, c'est une fonction $\frac{dx}{dt} = f(x)$, telle que x est l'état de n'importe quel système y compris multi-agent (SMA), et qui évolue en fonction du temps t de façon discrète ou continue.

Modèle Mirana : Un modèle de simulation compliqué et complexe permettant d'explorer et de discuter des scenarii de gestion intégrée des ressources naturelles renouvelables. Il a pour but

d'accompagner les modélisateurs et les différents acteurs du SES dans la démarche de modélisation d'accompagnement afin d'apprécier les impacts du transfert de gestion des ressources forestières par les VOI.

Observable : Une observable est une sortie du modèle de simulation qui peut être observée et/ou suivie au cours de la simulation.

Observation de modèle : Une observation est une spécification de trajectoires à partir des transformations successives calculées par la simulation et leur traitement pour extraire les indicateurs souhaités par les thématiciens.

Paramètre : Un paramètre est une donnée d'entrée du modèle fournie soit à partir du code informatique lui-même soit issue d'une source de données externe afin de construire plusieurs scénarii sur le même modèle.

Permis d'exploitation : Une autorisation administrative accordée à un exploitant en vue de prélever dans la forêt ou la parcelle forestière faisant l'objet du permis, un volume de bois déterminé pour approvisionner le marché national ou d'exportation.

Plan d'expériences : Une suite ordonnée d'essais d'une expérimentation permettant d'organiser au mieux les essais qui accompagnent une recherche scientifique. Son objectif principal est d'obtenir le maximum de renseignements avec le minimum d'expériences.

Population synthétique : Une population artificielle composée d'individus auxquels sont associées des caractéristiques individuelles (niveau n). Cette population est construite à partir des données connues au niveau agrégé n+1 du recensement. Le terme « synthétique » a été utilisé par Chris Barett [CL. Barrett, et al. 2004] dans le projet TRANSIMS pour parler d'une population de synthèse avec le principe de parcimonie, généré artificiellement par l'utilisateur.

Ressource Naturelle Renouvelable (RNR): Une ressource naturelle dont le stock peut se reconstituer sur une période courte à l'échelle humaine. Il faut que le stock puisse se renouveler au moins aussi vite qu'il est consommé.

Socio-écosystème (ou SES ou système socio-écologique) : Un système qui se compose d'unité «bio-géo-physique», de ses acteurs et des institutions sociales associées. C'est un système complexe et adaptatif, délimité par des limites spatiales ou fonctionnelles autour des écosystèmes particuliers.

Simulation : Une simulation peut être considérée comme la construction de la trajectoire d'états d'un modèle de simulation.

Système Multi-Agents (SMA) : Un système composé d'un ensemble d'agents artificiels ou naturels, situés dans un certain environnement et interagissant selon certaines relations pour produire des comportements.

Tavy : Une culture de riz pluvial sans labour sur défriche-brûlis de forêt naturelle humide.

Trajectoire : Une trajectoire est l'évolution de l'état d'un élément du modèle de simulation en fonction du temps.

Vondron'olona ifotony (VOI) : Une communauté locale de base

Annexes

Annexe A - Grammaires des langages dédiés

A.1. Grammaire du langage L1

```
grammar org.xtext.datainput.datasource.Datasource with
org.eclipse.xtext.common.Terminals

generate datasource "http://datainput/datasource/Datasource"
import"http://www.eclipse.org/emf/2002/Ecore"as ecore
import"http://www.eclipse.org/xtext/common/JavaVMTypes"as jvmTypes
ConnectionMetamodel returnsConnectionMetamodel:
	{ConnectionMetamodel}
	'DatasourceSpecification'
	'{'
		connectionSpecifications+=ConnectionSpecification*
	'}';
ConnectionSpecification returnsConnectionSpecification:
	DatabaseSpecification | CSVSpecification | XMLSpecification |
ExcelSpecification | ShapefileSpecification;

DatabaseSpecification returnsDatabaseSpecification:
	{DatabaseSpecification}
	'DatabaseSpecification'name=STRING
	'{'
		('dbtype'':' dbtype=STRING)?
		('user'':' user=STRING)?
		('passwd'':' passwd=STRING)?
		('host'':' host=STRING)?
		('port'':' port=STRING)?
		('schema'':' schema=STRING)?
		('database'':' database=STRING)?
	'}';
CSVSpecification returnsCSVSpecification:
	{CSVSpecification}
	'CSVSpecification'name=STRING
	'{'
		('url'':' url=STRING)?
		('fileName'':' fileName=STRING)?
		('encoding'':' encoding=STRING)?
		('fieldSeparator'':' fieldSeparator=STRING)?
		('rowSeparator'':' rowSeparator=STRING)?
		('headerRow'':' headerRow=Boolean)?
	'}';
XMLSpecification returnsXMLSpecification:
	{XMLSpecification}
	'XMLSpecification'name=STRING
	'{'
		('url'':' url=STRING)?
		('fileName'':' fileName=STRING)?
		('doctype'':' doctype=STRING)?
	'}';
ExcelSpecification returnsExcelSpecification:
	{ExcelSpecification}
	'ExcelSpecification'name=STRING
	'{'
		('url'':' url=STRING)?
```

```
            ('fileName'':' fileName=STRING)?
            ('headerRow'':' headerRow=Boolean)?
            ('sheet'':' sheet+=STRING*)?
      '}';
ShapefileSpecification returnsShapefileSpecification:
      {ShapefileSpecification}
      'ShapefileSpecification'name=STRING
      '{'
            ('url'':' url=STRING)?
            ('fileName'':' fileName=STRING)?
            ('spatialIndex'':' spatialIndex=Boolean)?
      '}';
grammar org.xtext.datainput.datastructure.Datastructure with
org.eclipse.xtext.common.Terminals

generate datastructure "http://www.xtext.org/datainput/datastructure/Datastructure"
import"http://www.eclipse.org/emf/2002/Ecore"as ecore
import"http://www.eclipse.org/xtext/common/JavaVMTypes"as jvmTypes
Model:
      elementtype+=ElementType*;
ElementType:
      {ElementType}
      'ElementType''{'
            ('name'name=STRING)?
            ('type' type=Type)?
            ('cardinality' cardinality=Cardinality)?
            ('annotation' annotation=ComplexType)?
      '}';
Cardinality:
      {Cardinality}
      ('[' min=(CardEnum)  '..'
          max=(CardEnum)
      ']')?
;
Type:
      type=ComplexType | type=SimpleType
;
SimpleType:
      {SimpleType}
      type=SimpleTypeEnum
;
ComplexType:
      {ComplexType}
      'ComplexType' elementtype+=ElementType*
;
enum CardEnum:
      ZERO ='0'|
      UN ='1'|
      MANY ='*'
  ;
enum SimpleTypeEnum:
      STRING = 'String'
      |INT= 'Integer'
      |ID='Id'
      |POINT='Point'
      |BLOB='Blob'
      |UUID='uuid'
      |MYBOOLEAN='Boolean'
      |MYDATE='Date'
```

```
      |EFloat='Float'
      |EInt = 'Long'
 ;
terminalHEX:
'a'..'h'|'A'..'H'|'0'..'9'
;
terminalMYDATE:
'0'..'9''0'..'9''0'..'9''0'..'9''-'
'0'..'9''0'..'9''-'
'0'..'9''0'..'9'
;
terminalUUID:
    HEX HEX HEX HEX HEX HEX HEX HEX '-'
    HEX HEX HEX HEX '-'
    HEX HEX HEX HEX '-'
    HEX HEX HEX HEX '-'
    HEX HEX HEX HEX HEX HEX HEX HEX HEX HEX HEX HEX
;
terminalBLOB:
'0' ('x'|'X') HEX+
;
terminalMYBOOLEANreturnsecore::EBoolean:
'true' | 'false' | 'TRUE' | 'FALSE'
;
```

A.2. Grammaire du langage L3

```
grammar org.xtext.description.model.Model with
org.eclipse.xtext.common.Terminals

generate model "http://www.xtext.org/description/model/Model"
import"http://www.eclipse.org/emf/2002/Ecore"as ecore
import"http://www.eclipse.org/xtext/common/JavaVMTypes"as jvmTypes

Modelreturns Model:
      {Model}
       concept+=Concept*;
Conceptreturns Concept:
      {Concept}
      'Concept'name=STRING 'timefunctionType' ft=TimeFunctionType '{'
            'represented by' path=STRING 'in' language = LanguageEnum ';'
            attribute+=Propriete*
      '}';
LanguageEnum:
      {LanguageEnum}
      name =
('Java'|'C++'|'C#'|'smalltalk'|'scheme'|'python'|'netlogo'|'GAMA')
;
Proprietereturns Propriete:
      {Propriete}
      'attribute' typetime = TimeFunctionType '{'
            ('name'name= STRING)';'
            ('type' type= STRING) ';'
      '}'
;
TimeFunctionTypereturns TimeFunctionType:
      'DifferenceEquation' type = DifferenceEquationType |
      'ConstantFunction' type = ConstantFunctionType |
      'ExplicitFunction' type = ExplicitFunctionType |
      'DifferentialEquation' type = DifferentialEquationType |
      'StepFunction' type = StepFunctionType |
      'Interpolation' type = InterpolationType
;
DifferenceEquationTypereturns DifferenceEquationType:
      'step' step= ID ';'
      constantParam += ConstantParameter*
;
ConstantParameterreturns ConstantParameter:
      'value' value= ID ';'
;
ConstantFunctionType returns ConstantFunctionType:
      constantParam = ConstantParameter
;
ExplicitFunctionTypereturns ExplicitFunctionType:
      {ExplicitFunctionType}
      constantParam+=ConstantParameter*
;
DifferentialEquationTypereturns DifferentialEquationType:
      {DifferentialEquationType}
      constantParam+=ConstantParameter*
;
StepFunctionTypereturns StepFunctionType:
      {StepFunctionType}
      'value' value+=ID* ';'
      'startTime' startTime = INT ';'
```

```
;
InterpolationTypereturnsInterpolationType:
      {InterpolationType}
      intervalleSequenceParameter += IntervalleSequenceParameter*
;
IntervalleSequenceParameterreturnsIntervalleSequenceParameter:
      'date' date=INT ';'
      'value' value=ID ';'
;
```

A.3. Grammaire du langage L2

```
grammarorg.xtext.L1toL3.L2with org.eclipse.xtext.common.Terminals

generate l2 "http://www.xtext.org/L1toL3/L2"
import"http://www.eclipse.org/emf/2002/Ecore"as ecore
import"http://www.eclipse.org/xtext/common/JavaVMTypes"as jvmTypes

ModelInit:
	'init'initializationName=ID';'
	initializers+=Initializer*;
Initializer:
	{Initializer}
	nameInitializer=ID '{'
	package+=Package*
		('transformation'transformation=Transformation)?
		('input''{'input+=Input*'}')?
		('output''{'output=Output?'}')?
	'}'
	;
Package:
	'package' packageName = STRING';'
;
Input:
	{Input}
	typeInput=ID nameInput = ID
	(('='datalist''(' dataListPathL1 = STRING ');')?|
	('='dataset''(' dataSetPathL1 = STRING ');')?|
	('='datalist''(' outputL3=ID ('.'method=Method)? ');')? |
	('='value=Type';')?)
;
Type:
	STRING | INT | ID
;
Output:
	{Output}
	'report'(nameOutput=ID)?(';')?
;
Transformation:
	{Transformation}'{'
	(method+= Method*)
	//(builder = Builder)?
	(switchCase=Switch)?'}'
;
Build:
	'new' constructor = ID'('param+=Parameter*')'
;
Switch:
	'switch''('objectTest=ID'){'(method=Method)? caseOf+=CaseOf* '}'
;
CaseOf:
	'case' valueTest=STRING ':' method=Method
;
Method:
	{Method}
	(typeObject=ID nameObject = ID'=')? ((('('cast=ID')')? object=ID
'.'methodCall=ID'('parameter+=Parameter*')')?|
	(build=Build)?)';'
;
Parameter:
	{Parameter} nameParam=ID;
```

A.4. Grammaire du langage L4

```
grammar org.xtext.modelquery.L4 with org.eclipse.xtext.common.Terminals

generate l4 "http://www.xtext.org/modelquery/L4"
Model:
	{Model}
	name=STRING
	tracking+=Tracking*
;
Tracking:
	{Tracking}
	'tracking'name=STRING '{'
	root = Root
	path = Path
	elementType+=ElementType*
	return = Return
	observationStrategy+=ObservationStrategy*
	'}'
;
Root:
	{Root}
	'root'':' root=STRING ';'
;
Path:
	{Path}
	'path'':' element+=Element* ';'
;
ElementType:
	{ElementType}
	'methodTracking' attribute=ID '{'
(getterSpecification=GetterSpecification |
		observationSpecification=ObservationSpecification
	)
	'}'
;
GetterSpecification:
	{GetterSpecification}

		('get'':' get=Get)?
		('getfor'':' getfor=GetFor)?
		('for'':'for=For)?
		('cast'':' cast=Cast)?

;
ObservationSpecification:
	{ObservationSpecification}
	'observerTracking'
	observedSpecification=ObservedSpecification
	listenerSpecification=ListenerSpecification
;
ObservedSpecification:
	{ObservedSpecification}
	'register' registerName=STRING
	'unRegister' unRegisterName=STRING
;
ListenerSpecification:
	{ListenerSpecification}
	'notify' notifyName=STRING
;
```

```
Get:
      {Get}
      methodCall=MethodCall
;
GetFor:
      {GetFor}
      methodCall=MethodCall
;
Cast:
      {Cast}
      '('class=ID')' attribute=ID ';'
;
MethodCall:
      {MethodCall}
      attribute=ID'.'method=ID'('param+=Parameter*')'';'
;
For:
      {For}
      attribute=ID ';'
;
Parameter:
      {Parameter}
      attribute=ID | attribute=STRING
;
Element:
      {Element}
      '/' attribute=ID ':' type=ID
;
Return:
      {Return}
      'return' nameReturn=ID '=' methodCall+=MethodCall | output+=ID
;
ObservationStrategy:
      {ObservationStrategy}
      'strategy'  ('sampling' samplingStrategy=SamplingStrategy | 'observing'
observerStrategy=ObserverStrategy)';'
;
ObserverStrategy:
      {ObserverStrategy}
      name=STRING'('output=ID')'
;
SamplingStrategy:
      {SamplingStrategy}
      time=Time
;
Time:
      {Time}
      ('singleTime''=' singleTime=SingleTime) | ('stepTime''=' stepTime=StepTime)
unit=Unit ';'
;

Unit:
      {Unit}
      unit = ('month' | 'year')
;

SingleTime:
      {SingleTime}
      singletime=INT
;
StepTime:    {StepTime}   steptime=INT;
```

A.5. Grammaire du langage L6

```
grammar org.xtext.output.L6 with org.eclipse.xtext.common.Terminals

generate l6 "http://www.xtext.org/output/L6"
OutputMetamodel:
	{OutputMetamodel}
	'LanguageType'':'languageChoice = LanguageEnum ';'
	'DatasinkSpecification' dataSinkSpecification+=DataSinkSpecification*
	;
DataSinkSpecification:
	'{'
		datasetSpecification=DataSetSpecification
		connectionSpecifications+=ConnectionSpecification*
	'}'
;
ConnectionSpecification returnsConnectionSpecification:
	GUISpecification |
	DatabaseSpecification |
	XMLSpecification |
	CSVSpecification |
	ExcelSpecification |
	ShapefileSpecification
	;
LanguageEnum:
	{LanguageEnum}
	name =
('Java'|'C++'|'C#'|'smalltalk'|'scheme'|'python'|'netlogo'|'GAMA')
;
DataSetSpecification returnsDataSetSpecification:
	{DataSetSpecification}
	'setModel' modelName=ID'.'setModelName=STRING ';'
;
DatabaseSpecification returnsDatabaseSpecification:
	{DatabaseSpecification}
	'DatabaseSpecification'name=STRING
	'{'
		('dbtype' dbtype=STRING ';')?('user' user=STRING ';')?('passwd'
passwd=STRING ';')?
		('host' host=STRING ';')?('port' port=STRING ';')?('schema'
schema=STRING ';')?
		('database' database=STRING ';')?
	'}';
GUISpecification returnsGUISpecification:
	{GUISpecification}
	'GUISpecification''{'
	guiSpecification = (PlotSpecification | TableSpecification |
MapSpecification | LabelSpecification)
	'}'
;
PlotSpecification:
	{PlotSpecification}
	'PlotSpecification'name=STRING '{'
'chartTitle' chartTitle=STRING ';'
'xLabel' x=INT ';'
'yLabel'y=INT ';'
'dataset' dataset=DataSetSpecification ';'
'}'
;
LabelSpecification returnsLabelSpecification:
	{LabelSpecification}
```

```
      'Title' labelTitle=STRING ';'
      'DataSet' dataset = DataSetSpecification ';'
;
TableSpecification:
      {TableSpecification}
      'Title' tableTitle=STRING ';'
      'DataSet' dataset = DataSetSpecification ';'
;
CSVSpecification returnsCSVSpecification:
      {CSVSpecification}
      'CSVSpecification'name=STRING
      '{'
            ('url'':' url=STRING)?
            ('fileName'':' fileName=STRING)?
            ('encoding'':' encoding=STRING)?
            ('fieldSeparator'':' fieldSeparator=STRING)?
            ('rowSeparator'':' rowSeparator=STRING)?
            ('headerRow'':' headerRow=Boolean)?
      '}';
XMLSpecification returnsXMLSpecification:
      {XMLSpecification}
      'XMLSpecification'name=STRING
      '{'
            ('url'':' url=STRING)?
            ('fileName'':' fileName=STRING)?
            ('doctype'':' doctype=STRING)?
      '}';
ExcelSpecification returnsExcelSpecification:
      {ExcelSpecification}
      'ExcelSpecification'name=STRING
      '{'
            ('url'':' url=STRING)?
            ('fileName'':' fileName=STRING)?
            ('headerRow'':' headerRow=Boolean)?
            ('sheet'':' sheet+=STRING*)?
      '}';
ShapefileSpecification returnsShapefileSpecification:
      {ShapefileSpecification}
      'ShapefileSpecification'name=STRING
      '{'
            ('url'':' url=STRING)?
            ('fileName'':' fileName=STRING)?
            ('spatialIndex'':' spatialIndex=Boolean)?
      '}';
      MapSpecification:
      {MapSpecification}
;
```

A.6. Grammaire du langage L5

```
grammarorg.xtext.L4toL6.L5with org.eclipse.xtext.common.Terminals

generate l5 "http://www.xtext.org/L4toL6/L5"
ModelInit:
	'indicator' indicatorName=ID';'
	indicatorBuilder+=IndicatorBuilder*;
IndicatorBuilder:
	{IndicatorBuilder}
	builderName=ID '{'
	package+=Package*
		('transformation'transformation=Transformation)?
		('input''{'input+=Input*'}')?
		('output''{'output=Output?'}')?
	'}'
	;
Package:
	'package' packageName = STRING';'
;
Transformation:
	{Transformation}'{'
	(methodSpecification+= MethodSpecification*)
;
MethodSpecification:
	{MethodSpecification}
	'language'':' languageName=STRING ';'
	'package'''packageName=Package';'
	method=Method';'
;
Method:
	{Method}
	(typeObject=ID nameObject = ID'=')? ((('('cast=ID')')? object=ID
'.'methodCall=ID'('parameter+=Parameter*')')?|
	(build=Build)?)';'
;
Input:
	{Input}
	typeInput=ID nameInput = ID
	(('=''datalist''(' dataListPathL1 = STRING ');')?|
	('=''dataset''(' dataSetPathL1 = STRING ');')?|
	('=''datalist''(' outputL3=ID ('.'method=MethodSpecification)? ');')? |
	('='value=Type';')?)
;
Type:
	STRING | INT | ID
;
Output:
	{Output}
	'report'(nameOutput=ID)?(';')?
;
```

Annexe B – Générateurs de code des langages dédiés

B.1. Générateur de code du langage L1

```
package org.xtext.datainput.datasource.generator

import org.eclipse.emf.ecore.resource.Resource
import org.eclipse.xtext.generator.IGenerator
import org.eclipse.xtext.generator.IFileSystemAccess

import org.xtext.datainput.datasource.datasource.ConnectionMetamodel
import org.xtext.datainput.datasource.datasource.ConnectionSpecification
import org.xtext.datainput.datasource.datasource.DatabaseSpecification
import org.xtext.datainput.datasource.datasource.XMLSpecification
import org.xtext.datainput.datasource.datasource.ExcelSpecification
import org.xtext.datainput.datasource.datasource.ShapefileSpecification
import org.xtext.datainput.datasource.datasource.CSVSpecification

/**
 * Generates code from your model files on save.
 *
 * see http://www.eclipse.org/Xtext/documentation.html#TutorialCodeGeneration
 */
class DatasourceGenerator implements IGenerator {

	overridevoid doGenerate(Resource resource, IFileSystemAccess fsa) {
		var connectionMetamodel = resource.contents.headas ConnectionMetamodel
		for (ConnectionSpecification
specification:connectionMetamodel.connectionSpecifications){
			fsa.generateFile("gen"+"/"+"mde"+"/"+"datasources"+"/"+ //package
  specification.name+".Java", specification.genSpecification)
		}
		fsa.generateFile("gen"+"/"+"mde"+"/"+"run"+"/"+ //package
"LoadDataStructureL1"+".Java", connectionMetamodel.genMain)
	}
	def genMain(ConnectionMetamodel connectionMetamodel)'''
		package gen.mde.run;
		import gen.mde.datasources.*;
		import jxl.read.biff.BiffException;
		import Java.io.IOException;
		import Java.io.BufferedWriter;
		import Java.io.FileWriter;
		import connection.specification.core.DataStructureExplorer;

		public class LoadDataStructureL1{
			DataStructureExplorer obs  = new DataStructureExplorer();
			public DataStructureExplorer getObs() {
				return obs;
			}
			public void setObs(DataStructureExplorer obs) {
				this.obs = obs;
			}

			public LoadDataStructureL1() throws IOException, BiffException{
				BufferedWriter fichier = new BufferedWriter(new
FileWriter("src-gen\\datastructure.dstruct"));
				DataStructureExplorer obs = new DataStructureExplorer();
				«FOR conn : connectionMetamodel.connectionSpecifications»
```

```
                        «conn.name»«conn.name.toFirstLower» = new
«conn.name»(this.getObs());
                        «ENDFOR»
                        //new LoadL1TreeView(this);
                    }

                    public static void main(String [] args) throws BiffException,
IOException{
                        new LoadDataStructureL1();
                    }
                }
        '''
        defdispatch genSpecification(DatabaseSpecification connectionSpec)'''
                package gen.mde.datasources;
                import connection.specification.core.DataBaseSpecification;
                import Java.io.IOException;
                import connection.specification.core.DataStructureExplorer;

                public class «connectionSpec.name» extends DataBaseSpecification{
                    DataBaseSpecification  «connectionSpec.name.toFirstLower» = new
DataBaseSpecification();
                    public «connectionSpec.name»(DataStructureExplorer obs)throws
IOException{
                «connectionSpec.name.toFirstLower».setDbType("«connectionSpec.dbtype»");
        «connectionSpec.name.toFirstLower».setDataBase("«connectionSpec.database»");
                «connectionSpec.name.toFirstLower».setUser("«connectionSpec.user»");
                «connectionSpec.name.toFirstLower».setPasswd("«connectionSpec.passwd»");
                «connectionSpec.name.toFirstLower».setHost("«connectionSpec.host»");
                    «connectionSpec.name.toFirstLower».setPort(«connectionSpec.port»);
                «connectionSpec.name.toFirstLower».setSchema("«connectionSpec.schema»");
                    «connectionSpec.name.toFirstLower».getConnection();
        «connectionSpec.name.toFirstLower».database2elementType("src\\datastructure.dst
ruct");
                    «connectionSpec.name.toFirstLower».loadDataBaseTree(obs);
                    System.out.println("\n");
                    }
                }
        '''
        defdispatch genSpecification(XMLSpecification connectionSpec)'''
        '''
        defdispatch genSpecification(ExcelSpecification connectionSpec)'''
                package gen.mde.datasources;
                import connection.specification.core.ExcelSpecification;
                import Java.io.IOException;
                import jxl.read.biff.BiffException;
                import connection.specification.core.DataStructureExplorer;
                public class «connectionSpec.name» extends ExcelSpecification{
                    ExcelSpecification  «connectionSpec.name.toFirstLower» = new
ExcelSpecification();
                    public «connectionSpec.name»(DataStructureExplorer obs)throws
IOException, BiffException{
                    «connectionSpec.name.toFirstLower».setUrl("«connectionSpec.url»");
                «connectionSpec.name.toFirstLower».setContains("«connectionSpec.sheet»");
                    «connectionSpec.name.toFirstLower».excel2elementType("src-
gen\\datastructure.dstruct");
                    «connectionSpec.name.toFirstLower».loadDataExcelTree(obs);
                    System.out.println("...Done \n");
                    }
```

```
		}
	'''
	defdispatch genSpecification(ShapefileSpecification connectionSpec)'''
		package gen.mde.datasources;
		import connection.specification.core.ShapeFileSpecification;
		import Java.io.IOException;
		import connection.specification.core.DataStructureExplorer;
		public class «connectionSpec.name» extends ShapeFileSpecification{
			ShapeFileSpecification  «connectionSpec.name.toFirstLower» = new
ShapeFileSpecification();
			public «connectionSpec.name»(DataStructureExplorer obs)throws
IOException{
			«connectionSpec.name.toFirstLower».setUrl("«connectionSpec.url»");
		«connectionSpec.name.toFirstLower».getConnection();
			«connectionSpec.name.toFirstLower».shape2elementType("src-
gen\\datastructure.dstruct");
			«connectionSpec.name.toFirstLower».loadDataShapeTree(obs);
			System.out.println("\n");
			}
		}
	'''
	defdispatch genSpecification(CSVSpecification connectionSpec)'''
		package gen.mde.datasources;
		import connection.specification.core.CSVSpecification;
		import Java.io.IOException;
		import connection.specification.core.DataStructureExplorer;
		public class «connectionSpec.name» extends CSVSpecification{
			CSVSpecification  «connectionSpec.name.toFirstLower» = new
CSVSpecification();
			public «connectionSpec.name»(DataStructureExplorer obs)throws
IOException{
			«connectionSpec.name.toFirstLower».setUrl("«connectionSpec.url»");
		«connectionSpec.name.toFirstLower».setHeader(«connectionSpec.headerRow»);
	«connectionSpec.name.toFirstLower».setSeparator('«connectionSpec.rowSeparator»'
);
		«connectionSpec.name.toFirstLower».getConnection();
			«connectionSpec.name.toFirstLower».csv2elementType("src-
gen\\datastructure.dstruct");
			«connectionSpec.name.toFirstLower».loadDataCSVTree(obs);
			System.out.println("\n");
			}}'''}
```

B.2. Générateur de code du langage L2

```
/*
 * generated by Xtext
 */
package org.xtext.L1toL3.generator

import org.eclipse.emf.ecore.resource.Resource
import org.eclipse.xtext.generator.IGenerator
import org.eclipse.xtext.generator.IFileSystemAccess
import org.xtext.L1toL3.l2.ModelInit
import org.xtext.L1toL3.l2.Initializer
import org.xtext.L1toL3.l2.Input
import org.xtext.L1toL3.l2.Transformation
import Java.lang.invoke.MethodHandles.Lookup
/**
 * Generates code from your model files on save.
 *
 * see http://www.eclipse.org/Xtext/documentation.html#TutorialCodeGeneration
 */
class L2Generator implements IGenerator {

	overridevoid doGenerate(Resource resource, IFileSystemAccess fsa) {
	var modelInit = resource.contents.headas ModelInit
	    fsa.generateFile("gen"+"/"+"mde"+"/"+"run"+"/"+ //package
	      modelInit.initializationName.toFirstUpper+".Java",
modelInit.genCodeJava)
	}
	def genCodeJava(ModelInit modelInit)'''
	package gen.mde.run;
	import Java.io.IOException;
	import Java.util.ArrayList;
	import Java.util.List;
	import jxl.read.biff.BiffException;
	import connection.specification.core.DataStructureExplorer;
	import Java.util.HashMap;
	import Java.util.Map;
	import Java.math.*;
	«FOR initializer:modelInit.initializers»«FOR project:
initializer.package»
	import «project.packageName».*;«ENDFOR»
	«ENDFOR»
	public class «modelInit.initializationName.toFirstUpper» {
	      static DataStructureExplorer explore;
	      «FOR initializer:modelInit.initializers»
	            «FOR method:initializer.transformation.method»
	                  «IF method.build!=null»
	                  static
List<«method.build.constructor»>«initializer.output.nameOutput» = new
ArrayList<«method.build.constructor»>();
	            «ENDIF»
	            «ENDFOR»
	      «ENDFOR»
	      public «modelInit.initializationName.toFirstUpper»
(DataStructureExplorer explorer){
	            explore = explorer;
	      }
	      public static List<Object> inputSet(String... dataPath){
	            ArrayList<Object> input = new ArrayList<Object>();
	            input.addAll(explore.dataWithoutDuplicationOf(dataPath));
	            return input;
	      }
```

```
	public static List<Object> inputList(String... dataPath){
		ArrayList<Object> input = new ArrayList<Object>();
		input.addAll(explore.dataWithDuplicationOf(dataPath));
		return input;
	}
	«FOR initializer:modelInit.initializers»
	«genMethodJava(initializer)»
	«ENDFOR»
	«modelInit.genMainMethodJava()»
}
'''
def genMainMethodJava(ModelInit modelInit) '''
public static void main(String [] args) throws BiffException,
IOException, Exception{
	new «modelInit.initializationName.toFirstUpper» (new
LoadDataStructureL1().getObs());
		«FOR initializer:modelInit.initializers»
			«initializer.nameInitializer»();
			«IF initializer.output!=null»

System.out.println(«initializer.output.nameOutput»);
			«ELSE»System.out.println("Without output");
			«ENDIF»
		«ENDFOR»
	}
'''
def genMethodJava(Initializer init)'''
public static void «init.nameInitializer»()throws Exception{
	«FOR input: init.input»
		«genInput(input)»
	«ENDFOR»
	«genTransformation(init)»
	}
'''
def genTransformation(Initializer init)'''
«IF init.transformation.method!=null&&
init.transformation.switchCase==null»
	«genMethods(init)»
«ELSEIF init.transformation.switchCase!=null»
	«genSwitchCase(init)»
«ENDIF»
'''
def genSwitchCase(Initializer initializer)'''
for (int i = 0; i <«initializer.input.get(0).nameInput».size(); i++){
	«genSwitchTopMethod(initializer)»
	«FOR caseOf:initializer.transformation.switchCase.caseOf»
		if
(«initializer.transformation.switchCase.objectTest».get(i).toString().equals("«
caseOf.valueTest»")) «caseOf.method.object».«caseOf.method.methodCall»(«FOR
param:caseOf.method.parameter»
	«FOR input:initializer.input»
		«IF input.nameInput == param.nameParam && input.value==null»
			«IF input.typeInput == 'String'»
				«param.nameParam.concat(".get(i).toString()")»«IF
param != caseOf.method.parameter.get(caseOf.method.parameter.size - 1)»,«ENDIF»
			«ELSEIF input.typeInput == 'Integer'»

Integer.parseInt(«param.nameParam.concat(".get(i).toString()")»)«IF param
!= caseOf.method.parameter.get(caseOf.method.parameter.size - 1)»,«ENDIF»
			«ELSEIF input.typeInput == 'Float'»
```

```
            Float.parseFloat(«param.nameParam.concat(".get(i).toString()")»)«IF
param != caseOf.method.parameter.get(caseOf.method.parameter.size - 1)»,«ENDIF»
                        «ELSEIF input.typeInput == 'double'»

      Double.parseDouble(«param.nameParam.concat(".get(i).toString()")»)«IF
param != caseOf.method.parameter.get(caseOf.method.parameter.size - 1)»,«ENDIF»
                        «ELSE»

      («input.typeInput»)«param.nameParam.concat(".get(i)")»«IF param !=
caseOf.method.parameter.get(caseOf.method.parameter.size - 1)»,«ENDIF»
                        «ENDIF»
                  «ENDIF»
                  «IF input.nameInput == param.nameParam && input.value!=null»
                        «IF input.typeInput == 'String'»
                              «param.nameParam.concat(".toString()")»«IF param
!= caseOf.method.parameter.get(caseOf.method.parameter.size - 1)»,«ENDIF»
                        «ELSEIF input.typeInput == 'Integer'»

      Integer.parseInt(«param.nameParam.concat(".toString()")»)«IF param !=
caseOf.method.parameter.get(caseOf.method.parameter.size - 1)»,«ENDIF»
                        «ELSEIF input.typeInput == 'Float'»

      Float.parseFloat(«param.nameParam.concat(".toString()")»)«IF param !=
caseOf.method.parameter.get(caseOf.method.parameter.size - 1)»,«ENDIF»
                        «ELSEIF input.typeInput == 'double'»

      Double.parseDouble(«param.nameParam.concat(".toString()")»)«IF param !=
caseOf.method.parameter.get(caseOf.method.parameter.size - 1)»,«ENDIF»
                        «ELSE»
                              («input.typeInput»)«param.nameParam»«IF param !=
caseOf.method.parameter.get(caseOf.method.parameter.size - 1)»,«ENDIF»
                        «ENDIF»
                  «ENDIF»
                  «ENDFOR»
            «ENDFOR»);
      «ENDFOR»
      }
      '''
      def genMethods(Initializer initializer)'''
      for (int i = 0; i <«initializer.input.get(0).nameInput».size(); i++){
      // Test if data is null...
      if(!«initializer.input.get(0).nameInput».get(i).toString().equals("")){

      «FOR method:initializer.transformation.method»
            «IF method.typeObject!=null&& method.nameObject!=null&&
method.build==null»
                  «method.typeObject»«method.nameObject» =
            «ENDIF»
            «IF method.cast!=null»(«method.cast»)«ENDIF»
            «IF method.object!=null&& method.methodCall!=null»
                  «method.object».«method.methodCall»«ENDIF»
            «IF
method.build!=null»«method.build.constructor»«method.build.constructor.toFirstL
ower» = new «method.build.constructor»«ENDIF»
            «IF method.build==null»
                  («FOR param:method.parameter»
                        «FOR input:initializer.input»
                              «IF input.nameInput == param.nameParam &&
input.value==null»
```

```
            «IF input.typeInput == 'String'»
        «param.nameParam.concat(".get(i).toString()")»«IF param !=
method.parameter.get(method.parameter.size - 1)»,«ENDIF»
                                    «ELSEIF input.typeInput == 'Integer'»
        Integer.parseInt(«param.nameParam.concat(".get(i).toString()")»)«IF param
!= method.parameter.get(method.parameter.size - 1)»,«ENDIF»
                                    «ELSEIF input.typeInput == 'Float'»
        Float.parseFloat(«param.nameParam.concat(".get(i).toString()")»)«IF param
!= method.parameter.get(method.parameter.size - 1)»,«ENDIF»
                                    «ELSEIF input.typeInput == 'double'»
        Double.parseDouble(«param.nameParam.concat(".get(i).toString()")»)«IF
param != method.parameter.get(method.parameter.size - 1)»,«ENDIF»
                                    «ELSE»
        («input.typeInput»)«param.nameParam.concat(".get(i)")»«IF param !=
method.parameter.get(method.parameter.size - 1)»,«ENDIF»

                                    «ENDIF»
                                «ENDIF»
                                «IF input.nameInput == param.nameParam &&
input.value!=null»
                                    «IF input.typeInput == 'String'»
        «param.nameParam.concat(".toString()")»«IF param !=
method.parameter.get(method.parameter.size - 1)»,«ENDIF»
                                    «ELSEIF input.typeInput == 'Integer'»
        Integer.parseInt(«param.nameParam.concat(".toString()")»)«IF param !=
method.parameter.get(method.parameter.size - 1)»,«ENDIF»
                                    «ELSEIF input.typeInput == 'Float'»
        Float.parseFloat(«param.nameParam.concat(".toString()")»)«IF param !=
method.parameter.get(method.parameter.size - 1)»,«ENDIF»
                                    «ELSEIF input.typeInput == 'double'»
        Double.parseDouble(«param.nameParam.concat(".toString()")»)«IF param !=
method.parameter.get(method.parameter.size - 1)»,«ENDIF»
                                    «ELSE»
        («input.typeInput»)«param.nameParam»«IF param !=
method.parameter.get(method.parameter.size - 1)»,«ENDIF»

                                    «ENDIF»
                                «ENDIF»
                            «ENDFOR»
                        «FOR methodHere:initializer.transformation.method»
                            «IF methodHere.nameObject == param.nameParam»
                                «IF methodHere.typeObject == 'String'»
                                    «param.nameParam.concat(".toString()")»«IF
param != method.parameter.get(method.parameter.size - 1)»,«ENDIF»
                                «ELSEIF methodHere.typeObject == 'Integer'»
        Integer.parseInt(«param.nameParam.concat(".toString()")»)«IF param !=
method.parameter.get(method.parameter.size - 1)»,«ENDIF»
                                «ELSEIF methodHere.typeObject == 'Float'»
        Float.parseFloat(«param.nameParam.concat(".toString()")»)«IF param !=
method.parameter.get(method.parameter.size - 1)»,«ENDIF»
```

```
            «ELSEIF methodHere.typeObject == 'double'»

      Double.parseDouble(«param.nameParam.concat(".toString()")»)«IF param !=
method.parameter.get(method.parameter.size - 1)»,«ENDIF»
                              «ELSE»

      («methodHere.typeObject»)«param.nameParam»«IF param !=
method.parameter.get(method.parameter.size - 1)»,«ENDIF»

                              «ENDIF»
                        «ENDIF»
                  «ENDFOR»
            «ENDFOR»);
      «ENDIF»
      «IF method.build!=null»
            («FOR param:method.build.param»
                  «FOR input:initializer.input»
                        «IF input.nameInput == param.nameParam &&
input.value==null»
                              «IF input.typeInput == 'String'»

      «param.nameParam.concat(".get(i).toString()")»«IF param !=
method.build.param.get(method.build.param.size - 1)»,«ENDIF»
                              «ELSEIF input.typeInput == 'Integer'»

      Integer.parseInt(«param.nameParam.concat(".get(i).toString()")»)«IF param
!= method.build.param.get(method.build.param.size - 1)»,«ENDIF»
                              «ELSEIF input.typeInput == 'Float'»

      Float.parseFloat(«param.nameParam.concat(".get(i).toString()")»)«IF param
!= method.build.param.get(method.build.param.size - 1)»,«ENDIF»
                              «ELSEIF input.typeInput == 'double'»

      Double.parseDouble(«param.nameParam.concat(".get(i).toString()")»)«IF
param != method.build.param.get(method.build.param.size - 1)»,«ENDIF»
                              «ELSE»

      («input.typeInput»)«param.nameParam.concat(".get(i)")»«IF param !=
method.build.param.get(method.build.param.size - 1)»,«ENDIF»

                              «ENDIF»
                        «ENDIF»
                        «IF input.nameInput == param.nameParam &&
input.value!=null»
                              «IF input.typeInput == 'String'»
                                    «param.nameParam.concat(".toString()")»«IF
param != method.build.param.get(method.build.param.size - 1)»,«ENDIF»
                              «ELSEIF input.typeInput == 'Integer'»

      Integer.parseInt(«param.nameParam.concat(".toString()")»)«IF param !=
method.build.param.get(method.build.param.size - 1)»,«ENDIF»
                              «ELSEIF input.typeInput == 'Float'»

      Float.parseFloat(«param.nameParam.concat(".toString()")»)«IF param !=
method.build.param.get(method.build.param.size - 1)»,«ENDIF»
                              «ELSEIF input.typeInput == 'double'»

      Double.parseDouble(«param.nameParam.concat(".toString()")»)«IF param !=
method.build.param.get(method.build.param.size - 1)»,«ENDIF»
                              «ELSE»
```

```
            («input.typeInput»)«param.nameParam»«IF param !=
method.build.param.get(method.build.param.size - 1)»,«ENDIF»
                                    «ENDIF»
                              «ENDIF»
                        «ENDFOR»
                        «FOR methodHere:initializer.transformation.method»
                              «IF methodHere.nameObject == param.nameParam»
                                    «IF methodHere.typeObject == 'String'»
                                          «param.nameParam.concat(".toString()")»«IF
param != method.build.param.get(method.build.param.size - 1)»,«ENDIF»
                                    «ELSEIF methodHere.typeObject == 'Integer'»

      Integer.parseInt(«param.nameParam.concat(".toString()")»)«IF param !=
method.build.param.get(method.build.param.size - 1)»,«ENDIF»
                                    «ELSEIF methodHere.typeObject == 'Float'»

      Float.parseFloat(«param.nameParam.concat(".toString()")»)«IF param !=
method.build.param.get(method.build.param.size - 1)»,«ENDIF»
                                    «ELSEIF methodHere.typeObject == 'double'»

      Double.parseDouble(«param.nameParam.concat(".toString()")»)«IF param !=
method.build.param.get(method.build.param.size - 1)»,«ENDIF»
                                    «ELSE»

      («methodHere.typeObject»)«param.nameParam»«IF param !=
method.build.param.get(method.build.param.size - 1)»,«ENDIF»
                                    «ENDIF»
                              «ENDIF»
                        «ENDFOR»
                  «ENDFOR»);

      «initializer.output.nameOutput».add(«method.build.constructor.toFirstLowe
r»
            );
            «ENDIF»
      «ENDFOR»
            }//end if
      }
      '''

      def genSwitchTopMethod(Initializer initializer)'''
      «IF initializer.transformation.switchCase.method.typeObject!=null&&
initializer.transformation.switchCase.method.nameObject!=null»

      «initializer.transformation.switchCase.method.typeObject»«initializer.tra
nsformation.switchCase.method.nameObject» =
      «ENDIF»
      «initializer.transformation.switchCase.method.object».«initializer.transf
ormation.switchCase.method.methodCall»(«FOR
param:initializer.transformation.switchCase.method.parameter»
      «FOR input:initializer.input»
            «IF input.nameInput == param.nameParam && input.value==null»
                  «IF input.typeInput == 'String'»
                        «param.nameParam.concat(".get(i).toString()")»«IF param
!=
initializer.transformation.switchCase.method.parameter.get(initializer.transfor
mation.switchCase.method.parameter.size - 1)»,«ENDIF»
                  «ELSEIF input.typeInput == 'Integer'»
```

```
            Integer.parseInt(«param.nameParam.concat(".get(i).toString()")»)«IF
param !=
initializer.transformation.switchCase.method.parameter.get(initializer.transfor
mation.switchCase.method.parameter.size - 1)»,«ENDIF»
                  «ELSEIF input.typeInput == 'Float'»

      Float.parseFloat(«param.nameParam.concat(".get(i).toString()")»)«IF param
!=
initializer.transformation.switchCase.method.parameter.get(initializer.transfor
mation.switchCase.method.parameter.size - 1)»,«ENDIF»
                  «ELSEIF input.typeInput == 'double'»

      Double.parseDouble(«param.nameParam.concat(".get(i).toString()")»)«IF
param !=
initializer.transformation.switchCase.method.parameter.get(initializer.transfor
mation.switchCase.method.parameter.size - 1)»,«ENDIF»
                  «ELSE»

      («input.typeInput»)«param.nameParam.concat(".get(i)")»«IF param !=
initializer.transformation.switchCase.method.parameter.get(initializer.transfor
mation.switchCase.method.parameter.size - 1)»,«ENDIF»

                  «ENDIF»
            «ENDIF»
            «IF input.nameInput == param.nameParam && input.value!=null»
                  «IF input.typeInput == 'String'»
                        «param.nameParam.concat(".toString()")»«IF param !=
initializer.transformation.switchCase.method.parameter.get(initializer.transfor
mation.switchCase.method.parameter.size - 1)»,«ENDIF»
                  «ELSEIF input.typeInput == 'Integer'»

      Integer.parseInt(«param.nameParam.concat(".toString()")»)«IF param !=
initializer.transformation.switchCase.method.parameter.get(initializer.transfor
mation.switchCase.method.parameter.size - 1)»,«ENDIF»
                  «ELSEIF input.typeInput == 'Float'»

      Float.parseFloat(«param.nameParam.concat(".toString()")»)«IF param !=
initializer.transformation.switchCase.method.parameter.get(initializer.transfor
mation.switchCase.method.parameter.size - 1)»,«ENDIF»
                  «ELSEIF input.typeInput == 'double'»

      Double.parseDouble(«param.nameParam.concat(".toString()")»)«IF param !=
initializer.transformation.switchCase.method.parameter.get(initializer.transfor
mation.switchCase.method.parameter.size - 1)»,«ENDIF»
                  «ELSE»
                        («input.typeInput»)«param.nameParam»«IF param !=
initializer.transformation.switchCase.method.parameter.get(initializer.transfor
mation.switchCase.method.parameter.size - 1)»,«ENDIF»

                  «ENDIF»
            «ENDIF»
      «ENDFOR»
      «ENDFOR»);
      '''

      def genInput(Input input)'''
      «IF input.dataListPathL1!=null»
            List<Object>«input.nameInput» =
inputList("«input.dataListPathL1.replace(".","\",\"")»");
            System.out.println(«input.nameInput»);
      «ENDIF»
```

```
	«IF input.dataSetPathL1!=null»
		List<Object>«input.nameInput» =
inputSet("«input.dataSetPathL1.replace(".","\",\"")»");
		System.out.println(«input.nameInput»);
	«ENDIF»
	«IF input.value!=null»
		«input.typeInput»«input.nameInput» = «input.value»;
	«ENDIF»
	'''
}
```

B.3. Générateur de code du langage L4

```
/*
 * generated by Xtext
 */
package org.xtext.modelquery.generator

import org.eclipse.emf.ecore.resource.Resource
import org.eclipse.xtext.generator.IGenerator
import org.eclipse.xtext.generator.IFileSystemAccess
import org.xtext.modelquery.l4.Model
import java.util.ArrayList
import java.util.List
import org.xtext.modelquery.l4.Output
import org.xtext.modelquery.l4.Return

/**
 * Generates code from your model files on save.
 *
 * see http://www.eclipse.org/Xtext/documentation.html#TutorialCodeGeneration
 */
class L4Generator implements IGenerator {
    override void doGenerate(Resource resource, IFileSystemAccess fsa) {
        var model = resource.contents.head as Model
        fsa.generateFile("gen"+"/"+"mde"+"/"+"L4"+"/"+ //package
        model.name.toFirstUpper+"Dynamics.java", model.genTracking)
    }

def dispatch genTracking(Model model)'''
package gen.mde.L4;

public class «model.name.toFirstUpper»Dynamics extends DefaultDynamics {
    /* Class variables */
    private static Logger log = Logger.getLogger(
    «model.name.toFirstUpper»Dynamics.class);
    «FOR tracking:model.tracking»
    private static void «tracking.name»(State state)throws EntityException{
    System.err.println("Printing «tracking.name»");
    log.debug("Dumping «tracking.name»...");
    «FOR methodTracking:tracking.elementType»
    «FOR elementPath:tracking.path.element»
    «IF i < tracking.path.element.size-1»
        «IF elementPath.attribute.toString.equals(
        methodTracking.attribute.toString)»
            «IF methodTracking.getterSpecification.cast != null &&
            methodTracking.getterSpecification.getfor == null»
            «methodTracking.getterSpecification.cast.class_»
            «methodTracking.getterSpecification.cast.attribute»  =
            («methodTracking.getterSpecification.cast.class_»)
            «methodTracking.attribute»;
            «ENDIF»
```

```
        «IF methodTracking.getterSpecification.getfor != null &&
        methodTracking.getterSpecification.cast == null»
        for («tracking.path.element.get(i+1).type»
        «tracking.path.element.get(i+1).attribute» :
            «IF i == 0» Root.get«elementPath.type.toString»(
            «FOR parameter:methodTracking.getterSpecification.
            getfor.methodCall.param»
            "«parameter.attribute.toString»"«ENDFOR»)){
        «ENDIF»
            «IF i>0»
            «methodTracking.getterSpecification.getfor.methodCall.attribute».
            «methodTracking.getterSpecification.getfor.methodCall.method»
            («FOR parameter:methodTracking.getterSpecification.
            getfor.methodCall.param»
            "«parameter.attribute.toString»"
            «ENDFOR»))
            {
    «ENDIF»
«ENDIF»
«ENDIF»
«ENDFOR»
«ENDFOR»


System.err.println("  "+«FOR return_:tracking.^return.element»
«IF return_.out!=null &&
return_.method==null»«return_.out»«ENDIF»
«IF return_.out==null && return_.method!=null»
«return_.method.attribute».«return_.method.method»(
«FOR param:return_.method.param»«param.attribute»
«ENDFOR»)
«ENDIF»+" "«IF iter < tracking.^return.element.size»
+/*«iter++»*/«ENDIF»«ENDFOR»);
state.sendProbe("«tracking.name»",new Object[]{
    «FOR return_:tracking.^return.element»
«IF return_.out!=null && return_.method==null»
«return_.out»«ENDIF»
«IF return_.out==null && return_.method!=null»
«return_.method.attribute».«return_.method.method»(
«FOR param:return_.method.param»«param.attribute»
«ENDFOR»)
«ENDIF»
«IF iter < tracking.^return.element.size»,
«ENDIF»
«ENDFOR»});
}«ENDFOR»
log.debug("...done.");
}
«ENDFOR»
/**
 * @see mimosa.dynamics.DefaultDynamics#doGetInternal(mimosa.state.State
 */
@Override
public void doGetInternal(State state) throws EntityException {
    «FOR tracking:model.tracking»
        «tracking.name» (state);
    «ENDFOR»
```

```
        // observation strategy HERE!!!
        «FOR time:model.tracking»state.sendInternal(
        «IF time.time.stepTime==null && time.time.singleTime!=null»
        «time.time.singleTime.singletime»
        «ENDIF»
        «IF time.time.stepTime!=null && time.time.singleTime==null»
        «time.time.stepTime.steptime»
        «ENDIF», "next");
        «ENDFOR»
    }
}
'''
}
```

B.4. Générateur de code du langage L5

```
 * generated by Xtext
package org.xtext.L4toL6.generator

import org.eclipse.emf.ecore.resource.Resource
import org.eclipse.xtext.generator.IGenerator
import org.eclipse.xtext.generator.IFileSystemAccess
import org.xtext.L4toL6.l5.ModelIndicator

override void doGenerate(Resource resource, IFileSystemAccess fsa) {
    var model = resource.contents.head as ModelIndicator
    fsa.generateFile("gen"+"/"+"mde"+"/"+"L5"+"/"+ //package
    model.indicatorProjectName.toFirstUpper+"Indicator.java", model.genAnalysis)
}

def CharSequence genAnalysis(ModelIndicator indicator)'''
  public class «indicator.indicatorProjectName.toFirstUpper»Indicator extends JFrame implements ActionListener {

      «FOR indicatorBuilder: indicator.indicatorBuilder»
      «FOR ifCount:indicatorBuilder.transformation.^if»
       //«var i = 0»
      «FOR condition: ifCount.condition»
      public static XYSeries serie«indicatorBuilder.builderName.toFirstUpper»«i» = new XYSeries("???");
      //«i++»
      «ENDFOR»
      «ENDFOR»

       «ENDFOR»


      // add components
      «FOR indicatorBuilder:indicator.indicatorBuilder»
      private JButton «indicatorBuilder.builderName»Analysis;
      private JTextField «indicatorBuilder.builderName.toFirstLower»;
      «ENDFOR»
      private JButton quit;
      /**
       * @throws HeadlessException
       */
      public «indicator.indicatorProjectName.toFirstUpper»Indicator() throws HeadlessException {
          super();
          setTitle("Analysis");
          Container content = this.getContentPane();
          content.add(splitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT));
          JPanel controlPanel = new JPanel();
          controlPanel.setLayout(new GridBagLayout());
          panelConstraint = new GridBagConstraints();
          panelConstraint.fill = GridBagConstraints.BOTH;
          panelConstraint.weightx = 0.5;
          panelConstraint.weighty = 0.5;

          // Controls
          JPanel buttons = new JPanel();
          buttons.setLayout(new BoxLayout(buttons,BoxLayout.Y_AXIS));
          buttons.setBorder(BorderFactory.createTitledBorder(BorderFactory.createLineBorder(Color.DARK_GRAY),
                  "Commands"));

          «FOR indicatorBuilder:indicator.indicatorBuilder»
              «indicatorBuilder.builderName»Analysis = new JButton("«indicatorBuilder.builderName.toFirstUpper» analy
              «indicatorBuilder.builderName»Analysis.addActionListener(this);
              buttons.add(«indicatorBuilder.builderName»Analysis);
          «ENDFOR»
```

```
        quit = new JButton("Quit");
        quit.addActionListener(this);
        buttons.add(quit);
        «FOR indicatorBuilder:indicator.indicatorBuilder»
        tableDatabase.add(new JLabel("«indicatorBuilder.builderName.toFirstUpper»"));
        «indicatorBuilder.builderName.toFirstLower» = new JTextField(10);
        «indicatorBuilder.builderName.toFirstLower».setText(prefs.get("«indicatorBuilder.
        builderName.toFirstLower»", "T«indicatorBuilder.builderName.toFirstUpper»"));
        tableDatabase.add(«indicatorBuilder.builderName.toFirstLower»);
        «ENDFOR»
    private void getTables() {
        if (parameterTable==null)
            parameterTable = new HashMap<String,String>();
            «FOR indicatorBuilder: indicator.indicatorBuilder»
            parameterTable.put("«indicatorBuilder.builderName.toFirstUpper»",
            «indicatorBuilder.builderName.toFirstLower».getText());
            «ENDFOR»
    }
    /**
     * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(ActionEvent evt) {
        «FOR indicatorBuilder: indicator.indicatorBuilder»
        if (evt.getSource()==«indicatorBuilder.builderName.toFirstLower»Analysis) {
            «indicatorBuilder.builderName.toFirstLower»Analysis();
        }
        «ENDFOR»
        else if (evt.getSource()==quit)
            exit();
    }
    «FOR indicatorBuilder: indicator.indicatorBuilder»
    //«indicatorBuilder.builderName.toFirstUpper» presentation
    private void «indicatorBuilder.builderName.toFirstLower»Analysis() {
        if (previousComponent!=null)
            splitPane.remove(previousComponent);
        getDBParameters();
        getTables();
        splitPane.setRightComponent(previousComponent=create«
        indicatorBuilder.builderName.
        toFirstUpper»Panel(outputDBParameters,parameterTable));
        this.pack();
        this.setLocationRelativeTo(null);
    }
    «ENDFOR»
    «FOR indicatorBuilder: indicator.indicatorBuilder»
    public JPanel create«indicatorBuilder.builderName.toFirstUpper»
    Panel(DBParameters parameters,Map<String,String> tables) {
        load«indicatorBuilder.builderName.toFirstUpper»Table(parameters,tables);
    JFreeChart chart = create«indicatorBuilder.builderName.
        toFirstUpper»Chart(create«indicatorBuilder.builderName.
        toFirstUpper»Dataset());
    ChartPanel panel = new ChartPanel(chart);
    panel.setMouseWheelEnabled(true);
    return panel;
    }

    «ENDFOR»
      «FOR indicatorBuilder: indicator.indicatorBuilder»
       private static XYDataset create«indicatorBuilder.builderName.
       toFirstUpper»Dataset() {
            XYSeriesCollection dataset = new XYSeriesCollection();
            «FOR ifCount:indicatorBuilder.transformation.^if»
```

```
                //«var i = 0»
                «FOR condition: ifCount.condition»
                dataset.addSeries(serie«indicatorBuilder.builderName.toFirstUpper»«i»);
                //«i++»
                «ENDFOR»
            «ENDFOR»
            return dataset;
        }
    «ENDFOR»
    «FOR indicatorBuilder: indicator.indicatorBuilder»
    public static void load«indicatorBuilder.builderName.toFirstUpper»
    Table(DBParameters parameters,Map<String,String> tables) {
        System.out.println("Loading the «indicatorBuilder.
        builderName.toFirstUpper» tables");
        try {
            System.out.println("Opening connection");
            Connection dbConn = DriverManager.getConnection(
                    "jdbc:postgresql://"+parameters.hostName+
                    ":"+parameters.portNumber+"/"+parameters.tableName,
                    parameters.userName,
                    parameters.passWord);
            Statement stmt = dbConn.createStatement();
            System.out.println();

            System.out.println("Getting the «indicatorBuilder
            .builderName.toFirstUpper» records");
            String query = "select «FOR input : indicatorBuilder.input»
            \"«input.nameInput»\",«ENDFOR» from \""+tables.
            get("«indicatorBuilder.builderName.toFirstUpper»")+"\"";
            ResultSet rs = stmt.executeQuery(query);
            while (rs.next()) {
                «FOR ifCountains : indicatorBuilder.transformation.^if»
                //«var i = 0»
                «FOR condition:ifCountains.condition»
                if (rs.getString("«condition.lhs.toString»").equals(«condition.rhs»)){
                //if (rs.getString("subject").equals("constrained(1)")){
                «FOR input:indicatorBuilder.input»
                «input.typeInput» «input.nameInput» = rs.get«input.typeInput.
                toFirstUpper»("«input.nameInput»");
                «ENDFOR»
                serie«indicatorBuilder.builderName.toFirstUpper»«i».add(time, quantity);
                }
                //«i++»
                «ENDFOR»
                «ENDFOR»
            }
        }
        catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    «ENDFOR»
```

Annexe C - Implémentation du modèle ECEC

```
public class Eating extends Activity {
	/* Class variables */
	private static Logger log = Logger.getLogger(Eating.class);
	private static final Collection<String> LOCAL_OUT_MESSAGES =
Arrays.asList("get");
	/* Instance variables */
	private boolean first = true;

	/* As Activity */
	/**
	 * @see hina.activities.Activity#getOutputMessages()
	 */
	@Override
	public Collection<String> getOutputMessages() {
		return Activity.mergeStrings(super.getOutputMessages(),
LOCAL_OUT_MESSAGES);
	}
	/**
	 * @see faritra.activities.Activity#getNextDuration(faritra.agents.Subject)
	 */
	@Override
	public int getNextDuration(Subject subject) throws EntityException {
		if (first) {
			first = false;
			return 1;
		} else
			return 3;
	}
	/**
	 * @see faritra.activities.Activity#execute(faritra.agents.Subject)
	 */
	@Override
	public void execute(Subject subject) throws EntityException {
		try {
			ResourceHolder here = (ResourceHolder)
subject.getCurrentPosition();
			// who I am talking to
			final OrganizationPlace who = getOrganizationPlace("field");
			// the quantity of resource from the point of view of who I am
talking to
			double availableQuantity =
here.getQuantity(who.getResource("biomass"));
			// gets it

	get("field",getResource("biomass"),availableQuantity*(subject.getType().equals(
"constrained")?0.5:0.9));
			if (log.isDebugEnabled())
				log.debug(subject+" eats
"+availableQuantity*(subject.getType().equals("constrained")?0.5:0.9)+" of biomass.");
		} catch (UndefinedResourceException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

}
```

```
public class Growing extends Activity {

	/* Class variables */
	private static Logger log = Logger.getLogger(Growing.class);
	private static final Collection<String> LOCAL_TABLES =
Arrays.asList("GrowthTable");

	/* As Activity */
	/**
	 * @see hina.activities.Activity#getTables()
	 */
	@Override
	public Collection<String> getTables() {
		return Activity.mergeStrings(super.getTables(), LOCAL_TABLES);
	}
	/**
	 * @see faritra.activities.Activity#getNextDuration(faritra.agents.Subject)
	 */
	@Override
	public int getNextDuration(Subject subject) throws EntityException {
		return 3;
	}
	/**
	 * @see faritra.activities.Activity#execute(faritra.agents.Subject)
	 */
	@Override
	public void execute(Subject subject) throws EntityException {
		// The growth itself
		for (ConstitutiveNorm
spaceName:GrowthDescription.getSpaces(getOrganization())) {
			Space space = (Space) getOrganization().getDenotation(spaceName);
			Collection<GrowthDescription> rates =
GrowthDescription.getGrowth(getOrganization(),spaceName);
			if (log.isDebugEnabled())
				log.debug(subject+" applies growth in "+space+" according to
"+rates);
			if (space!=null && rates!=null)
				for (Place place:space)
growResources(place,rates,spaceName);
		}
	}

	/**
	 * Grows the resources on a given place of a space, given the growing rates
	 * @param place The place
	 * @param rates The growing rates
	 * @param spaceName The space name.
	 */
	private void growResources(Place place,Collection<GrowthDescription>
rates,ConstitutiveNorm spaceName) {
		ResourceHolder resourceHolder = (ResourceHolder) place;
		for (GrowthDescription rate : rates) {
			Resource resource = rate.resource;
			double quantity = resourceHolder.getQuantity(resource);
			if (quantity > 0) {
				double increase = quantity * rate.growthRate;
				if (rate.max > 0 && place instanceof GeographicPlace) {
					double limit;
					limit = 1-quantity/rate.max;
```

```
		resourceHolder.addResource(rate.resource,increase*limit);
                        } else resourceHolder.addResource(rate.resource,increase);
                }
        }
	}

}
public class Living extends AgentManagement {

	/* Class variables */
	private static Logger log = Logger.getLogger(Living.class);

	/* As Activity */
	/**
	 * @see faritra.activities.Activity#getNextDuration(faritra.agents.Subject)
	 */
	@Override
	public int getNextDuration(Subject subject) throws EntityException {
		return 3;
	}
	/**
	 * @see faritra.activities.Activity#execute(faritra.agents.Subject)
	 */
	@Override
	public void execute(Subject subject) throws EntityException {
		// gets the harvested biomass
		double harvest = subject.getQuantity(getResource("biomass"));
		subject.removeResource(getResource("biomass"));
		// adds to the energy
		subject.addResource(getResource("energy"), harvest);
		if (log.isDebugEnabled())
			log.debug(subject+" converts "+harvest+" of biomass into energy.");
		// consume metabolic consumption
		subject.removeResource(getResource("energy"), 2);
		if (log.isDebugEnabled())
			log.debug(subject+" consumes "+2+" of energy remaining
"+subject.getQuantity(getResource("energy")));
		// check remaining energy
		double energy = subject.getQuantity(getResource("energy"));
		if (energy <= 0) {				// condition to die
			if (log.isDebugEnabled())
				log.debug(subject+" dies.");
			subject.die();
		} else if (energy >= 100) {
			// where I am
			Place here = subject.getCurrentPosition();
			// neighbors not occupied
			PathList neighbours = here.getNeighbourPaths();
			neighbours.select(new PlaceCondition() {
				@Override
				public boolean check(Place place) {
					return !((SubjectHolder)place).occupied();
				}
				@Override
				public boolean finished() {
					return false;
				}
			});
```

```
                        Path dest = neighbours.oneOf();
                        Subject newSubject = subject.reproduce((Position)
dest.lastPlace());
                        newSubject.removeResource(getResource("energy"));
                        newSubject.addResource(getResource("energy"), 50);
                        subject.removeResource(getResource("energy"), 50);
                        if (log.isDebugEnabled())
                                log.debug(subject+" gives birth to "+newSubject);
                }
        }

}
public class Moving extends Activity {

        /* Class variables */
        private static Logger log = Logger.getLogger(Moving.class);
        private static final Collection<String> LOCAL_OUT_MESSAGES =
Arrays.asList("move");
        /* Instance variables */
        private boolean first = true;

        /* As Activity */
        /**
         * @see hina.activities.Activity#getOutputMessages()
         */
        @Override
        public Collection<String> getOutputMessages() {
                return Activity.mergeStrings(super.getOutputMessages(),
LOCAL_OUT_MESSAGES);
        }
        /**
         * @see faritra.activities.Activity#getNextDuration(faritra.agents.Subject)
         */
        @Override
        public int getNextDuration(Subject subject) throws EntityException {
                if (first) {
                        first = false;
                        return 1;
                } else
                        return 3;
        }
        /**
         * @see faritra.activities.Activity#execute(faritra.agents.Subject)
         */
        @Override
        public void execute(Subject subject) throws EntityException {
                // where I am
                Place here = subject.getCurrentPosition();
                // neighbors...
                PathList neighbours = here.getNeighbourPaths();
                // ...not occupied...
                neighbours.select(new PlaceCondition() {
                        @Override
                        public boolean check(Place place) {
                                return !((SubjectHolder)place).occupied();
                        }
                        @Override
                        public boolean finished() {
                                return false;
```

```
                        }
                });
                // ...including here
                neighbours = neighbours.union(here);
                // who I am talking to
                final OrganizationPlace who = getOrganizationPlace("field");
                // gets the place with the maximum biomass
                Path to = neighbours.maxOf(new PlaceValue<Double>() {
                        public Double value(Place place) {
                                // gets the resource from the point of view of whom I am
talking to
                                try {
                                        return
((ResourceHolder)place).getQuantity(who.getResource("biomass"));
                                } catch (UndefinedResourceException e) {
                                        log.warn("Resource "+e.getResource()+" undefined in
"+who);
                                        return 0.0;
                                }
                        }
                });
                // ask to field to go there
                move("field",to);
                if (log.isDebugEnabled()) log.debug(subject+" moves to "+to);
        }

}
```

Table des matières