Une architecture logicielle pour la multi-modélisation et la simulation à évènements discrets de systèmes naturels complexes

Jean Baptiste Filippi

UNIVERSITÉ DE CORSE – PASQUALE PAOLI

U.F.R. SCIENCES ET TECHNIQUES

THÈSE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE CORSE

ÉCOLE DOCTORALE ENVIRONNEMENT ET SOCIÉTÉ

Discipline : Sciences pour l'Environnement

Spécialité : Informatique

présentée par

Jean Baptiste FILIPPI

Une architecture logicielle pour la multi-modélisation et la simulation à évènements discrets de systèmes naturels complexes

---

sous la direction du Professeur

Paul BISGAMBIGLIA

Présentée publiquement le 17 décembre 2003 devant le jury composé de :

Rapporteurs :	M. David HILL, *Professeur, Université Blaise Pascal*
	M. Bernard P. Zeigler, *Professeur, The University of Arizona*
Examinateurs :	M. Fernando Barros, *Professeur, Université de Coimbra*
	M. Paul-Antoine BISGAMBIGLIA, *Professeur, Université de Corse*
	M. Marco Di Natale, *Professeur, École supérieure Sainte Anne de Pise*
	M. Tahar Khammaci, *Maître de Conférences, Université de Nantes*
	M. Jean-François SANTUCCI, *Professeur, Université de Corse*

UNIVERSITÉ DE CORSE – PASQUALE PAOLI

U.F.R. SCIENCES ET TECHNIQUES

THÈSE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE CORSE

ÉCOLE DOCTORALE ENVIRONNEMENT ET SOCIÉTÉ

Discipline : Sciences pour l'Environnement

Spécialité : Informatique

présentée par

Jean Baptiste FILIPPI

Une architecture logicielle pour la multi-modélisation et la simulation à évènements discrets de systèmes naturels complexes

---

sous la direction du Professeur

Paul BISGAMBIGLIA

Présentée publiquement le 17 décembre 2003 devant le jury composé de :

Rapporteurs :	M. David HILL, *Professeur, Université Blaise Pascal*
	M. Bernard P. Zeigler, *Professeur, The University of Arizona*
Examinateurs :	M. Fernando Barros, *Professeur, Université de Coimbra*
	M. Paul-Antoine BISGAMBIGLIA, *Professeur, Université de Corse*
	M. Marco Di Natale, *Professeur, École supérieure Sainte Anne de Pise*
	M. Tahar Khammaci, *Maître de Conférences, Université de Nantes*
	M. Jean-François SANTUCCI, *Professeur, Université de Corse*

A mon père.

Remerciements

Cette thèse s'est déroulée au sein de l'équipe Modélisation Informatique du laboratoire Systèmes Physiques pour l'Environnement (Unité Mixte de Recherche CNRS 6134) de l'Université de Corse. Ce travail a été réalisé grâce au soutien financier de la Collectivité Territoriale de Corse. Que ces deux institutions trouvent ici le témoignage de ma reconnaissance.

Je tiens tout particulièrement à exprimer ma plus profonde gratitude à Monsieur Paul BISGAMBIGLIA, Professeur à l'Université de Corse, pour avoir dirigé ces travaux et m'avoir soutenu dans cette étude. Sa disponibilité et ses conseils ont été indispensables à la concrétisation de cette recherche.

À Monsieur David HILL, professeur à l'Université Blaise Pascal et Monsieur Bernard P. ZEIGLER, Professeur à l'Université de l'Arizona, j'adresse ma plus respectueuse reconnaissance pour l'intérêt qu'ils ont porté à ce travail en acceptant d'en être les rapporteurs dans ce jury.

Que Monsieur Fernando BARROS, Professeur à l'Université de Coimbra, Monsieur Marco DI NATALE, Professeur à l'école supérieure Sainte Anne de Pise, et Monsieur Tahar KHAMMACI, Maitre de conférences à l'Université de Nantes, qui me font l'honneur de participer à ce jury, trouvent ici l'expression de ma profonde gratitude.

Je voudrais également exprimer toute ma reconnaissance à Monsieur Jean-François SANTUCCI qui m'a accueilli dans son laboratoire, pour sa disponibilité et pour avoir accepté de participer à ce jury.

Je remercie tous les doctorants, chercheurs et personnels de l'Université de Corse pour leur grande gentillesse et leur contribution à l'élaboration de ce travail, avec une pensée particulière à Fabrice, Fernand, Jeff et Marielle et plus généralement aux voisins du second étage.

Ma plus profonde reconnaissance va à Marie-Laure et Christophe qui tous les jours pendant trois ans, m'ont donné des conseils précieux et beaucoup de leur temps pour élaborer ce travail.

Ma gratitude éternelle va à mes parents pour leur soutien sans faille durant toutes ces années d'études. Un grand merci à Michel, mon oncle, pour avoir passé des nuits à traquer les fautes dans ce manuscrit. J'exprime également ma gratitude à Véro pour son soutien actif et sa capacité à supporter les contraintes qui découlaient de ce travail. Finalement, un grand merci à tous mes amis qui m'ont été d'un soutien indispensable dans des moments quelquefois difficiles.

Table des matières

Introduction générale

Au début de toute discipline scientifique, les chercheurs construisent leurs propres outils d'expérimentations : soufflent leurs propres éprouvettes, sculptent leurs profils d'ailes, enroulent les fils de leurs propres détecteurs de particules, voire même fabriquent leurs propres ordinateurs. Ils doivent donc cumuler les fonctions de scientifique, d'ingénieur, de mécanicien et d'électricien. Une fois la discipline bien établie, la collaboration entre scientifiques et ingénieurs aboutit au développement d'équipements standards plus sûrs. Ces instruments permettent aux scientifiques de se concentrer sur leur recherche plutôt que sur la fabrication d'outils. L'utilisation d'outils standards n'obéit pas seulement à un souci de confort, elle permet de mettre en place des protocoles opératoires complémentaires, vérifiables et reproductibles.

La modélisation informatique est devenue aujourd'hui un outil essentiel pour l'étude de systèmes naturels complexes dans des disciplines variées pouvant être très éloignées de l'informatique : l'écologie, la biologie, la mécanique. Les chercheurs dans ces disciplines passent beaucoup de temps à construire isolément leurs propres appareillages expérimentaux logiciels, équivalents des bobines dans le domaine de l'informatique. Malheureusement, l'implémentation informatique de modèles transforme souvent de très bons scientifiques en de mauvais programmeurs. En conséquence, la plupart des outils expérimentaux apparaissent mal conçus d'un point de vue logiciel.

Les résultats obtenus à partir de ces différents outils se révèlent difficiles à comparer avec les

résultats d'autres expériences et difficiles à reproduire à cause des choix de conception faits dans ces logiciels spécifiques.

De plus, écrire un logiciel est généralement une perte de capital temps. Dans de nombreux cas, les mêmes fonctionnalités sont réécrites par divers groupes de recherche internationaux.

Enfin, les problèmes qui se posent dans le domaine de l'étude de systèmes naturels deviennent de plus en plus complexes et transdisciplinaires [Fishwick, 1995]. Leurs résolutions rendent nécessaire, pour les résoudre, la collaboration de chercheurs et de leurs outils de simulation. Or ces outils construits isolément se révèlent en général incompatibles.

Il existe donc un besoin manifeste d'harmonisation de méthodes et d'outils standards dans le domaine de la modélisation et de la simulation de systèmes naturels complexes. Les recherches menées dans notre laboratoire de modélisation informatique tendent à identifier les besoins d'un tel ensemble et à proposer des méthodes et outils qui permettront un meilleur échange entre chercheurs ainsi qu'une plus grande efficacité de conception.

Objectifs

Notre objectif principal est de fournir un outil évolutif de modélisation et de simulation de systèmes naturels. Pour cela nous allons proposer une architecture générique logicielle, plus précisément un cadriciel (framework), dans le souci d'offrir une réponse générique à notre problématique. L'originalité de l'approche envisagée est de permettre l'ajout ultérieur de techniques, à un cadriciel existant, par la voie de packages spécifiques. L'utilisation de packages permet de respecter la cohérence de l'architecture globale du cadriciel tout en assurant son évolutivité. De plus, grâce à l'utilisation d'un formalisme unificateur, DEVS [Zeigler, 2000], sur lequel devra se baser chaque technique ajoutée au cadriciel, il est possible d'assurer la compatibilité entre les différents modèles créés. Le caractère spécifique du domaine de l'étude de systèmes naturels impose aussi un certain nombre de contraintes de visualisation et de présentation de résultats que nous devrons résoudre. En ce sens, comme les Systèmes d'Information Géographiques sont très largement utilisés pour observer des données environnementales, ils constituent un outil qu'il est nécessaire d'intégrer

dans notre approche [Chiari et al., 2000b]. De même, nous prendrons en compte l'intégration dans le cadriciel de méthodes originales de modélisation et de simulation de phénomènes spatiaux.

Synoptique

Ce mémoire s'articule en six chapitres :

- Le premier chapitre aborde le cadre de notre problématique et présente les concepts clés de notre approche.
- Le second chapitre fait une revue des approches logicielles accessibles dans le domaine de la modélisation et de la simulation basé sur DEVS et dans le domaine de la simulation environnementale.
- Le troisième chapitre présente la partie principale de l'approche logicielle, composée des packages de base définissant l'interface graphique, le stockage de modèle, les cadres expérimentaux et le moteur de modélisation et de simulation.
- Le chapitre quatre propose les ajouts de techniques nécessaires à l'approche logicielle en vue de modélisation et de simulation.
- Le chapitre cinq expose l'implémentation logicielle de l'approche générique, JDEVS.
- Enfin, le chapitre six illustre à travers trois exemples d'expérimentation, la justification de notre approche.

L'intégralité des logiciels développés dans le cadre de cette thèse se trouve sous support CD-ROM en annexe 6.4.

CHAPITRE 1

Problématique

Ce chapitre expose les bases du problème à résoudre dans le cadre de cette étude. La première partie présente les concepts clés qui fondent la méthodologie de modélisation / simulation utilisée comme base de cette étude. Parmi ces concept clés, nous détaillerons plus particulièrement ceux de : système, modèle, simulation, expérimentation et validation. Nous porterons également notre attention sur les étapes de la construction d'un modèle et ce, sous les deux aspects complémentaires de la théorie de la modélisation et de la simulation que constituent, les niveaux et les formalismes de spécification d'un système. Le seconde partie sera consacrée aux méthodes de multi-modélisation qui nous permettent de définir un environnement général de modélisation de systèmes dynamiques complexes. L'exposé de ces méthodes montrera qu'un cadre général unifié est nécessaire pour l'étude de ces systèmes. Ceci permettra de définir les points essentiels du problème à résoudre dans le domaine de compétence du laboratoire UMR CNRS 6134 : la modélisation de systèmes physiques pour l'environnement.

1.1 Théorie de la modélisation et de la simulation

La modélisation est l'établissement d'un modèle pour répondre à des questions. Cette définition, bien que très concise, résume l'ampleur de l'enjeu de notre analyse et pour une définition plus complète des modèles, nous pouvons nous référer à celle de [Minsky, 1968] :

Pour un opérateur O, un objet M est un modèle d'un objet A si O peut utiliser M pour répondre à des questions de A.

Le développement de la discipline de la modélisation est né de la problématique de la construction de tels modèles. La théorie de la modélisation et de la simulation s'est basée sur la théorie générale des systèmes [Wymore, 1977], pour proposer des formalismes et des méthodes permettant de décrire des systèmes.

1.1.1 Concepts généraux

La théorie générale des systèmes distingue la *structure* (constitution interne d'un système) du *comportement* (ses manifestations externes). Elle définit les modèles comme causaux (dont les sorties sont la conséquence d'une entrée) et déterministes (à une entrée donnée ne peut correspondre qu'une seule sortie). Cette théorie sert de base à bon nombre de formalismes tels les équations différentielles, les automates à états finis, les réseaux de Petri, *etc*. Sa forme générale est pour un système A :

$A \equiv < \tau, X, \Omega, S, Y, \delta, \lambda >$ avec :

τ : base de temps,

X : ensemble des états d'entrée,

$\Omega : \tau \rightarrow \chi$: états d'entrée courants,

S : ensemble des états du modèle,

$\delta : \Omega \times S \rightarrow S$: fonction, de transition (fait évoluer l'état du modèle en fonction d'activations),

Y : ensemble des états de sortie,

$\lambda : S \rightarrow Y$: fonction de sortie.

La base de temps τ est la formalisation de la variable indépendante de temps. L'ensemble des états d'entrée *X* constitue toutes les activations d'entrées possibles pour le système, chaque entrée étant associée à un intervalle de temps. *S* représente l'ensemble des états que peut prendre le modèle. La dynamique de ce système est décrite dans la fonction de transition, δ qui applique les états d'entrée courants $\Omega \in \Omega$ à l'état courant pour transiter vers un nouvel état. Le système peut générer une sortie obtenue en appliquant la fonction λ à partir de l'état courant.

Les fonctions de transition et de sortie sont activées pour étudier l'évolution d'un modèle pendant la simulation, mais comment simuler ? Grâce à la théorie des systèmes, les algorithmes de simulation développés pour une structure de modèle peuvent être utilisés pour tous les modèles utilisant cette structure. Nous nous intéresserons donc d'abord aux différents types de simulation de modèles avant de revenir à leurs méthodes spécifications.

1.1.2 La simulation comme mode d'expérimentation virtuelle

La simulation consiste à étudier l'évolution de l'état d'un modèle à travers le temps. Le temps, comme les états d'un système à analyser peut être vu comme continu ou discret. Selon que les états du système sont spécifiés de manière dénombrable ou non dans un modèle on parle de simulation discrète ou continue. Parmi ces systèmes, la distance séparant une planète de son étoile, ou la variation de température dans l'atmosphère constituent des systèmes à états continus car leurs valeurs changent continuellement. Un système de trafic urbain peut être vu au contraire comme un système à états discrets car l'arrivée de voitures à un carrefour est un événement subi et dont les quantités sont dénombrables. La figure 1.1 présente la simulation continue de l'évolution d'un modèle à travers le temps, ce type d'analyse nécessite une description de type mathématique analytique du modèle car il doit être possible de donner l'état du système en tout temps.

La complexité de tels modèles grandit toutefois avec le nombre de paramètres et il devient rapidement impossible de modéliser des systèmes complexes de manière purement analytique. Il est alors nécessaire de décrire ces systèmes dans des approches de simulation discrète. Dans une approche de simulation discrète, l'état futur du modèle dépend de son état actuel. Les méthodes de

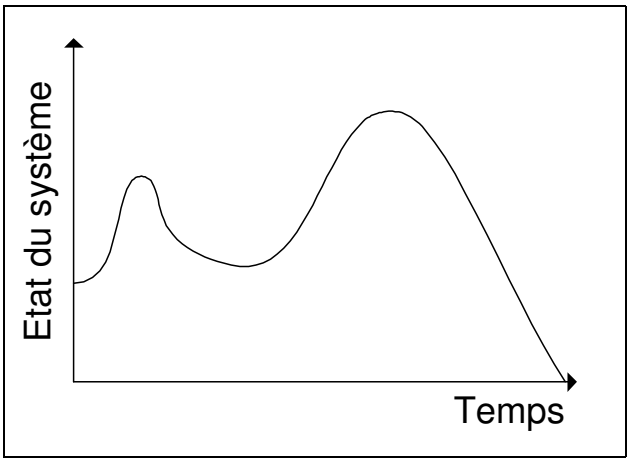

FIG. 1.1*: Simulation continue*

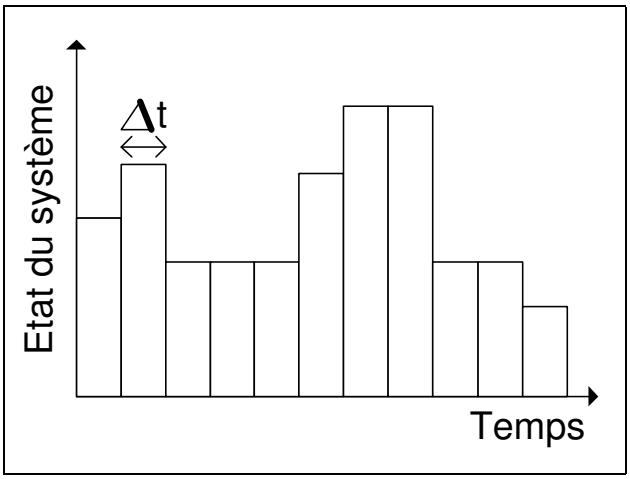

FIG. 1.2*: Simulation discrète, dirigée par horloge.*

simulation discrètes correspondent aux méthodes de résolution d'équations différentielles de type Euler ou Runge-Kutta [Hubbard, 1999].

La figure 1.2 représente une simulation discrète. La simulation de ce type de modèles implique que les changements d'états s'effectuent de manière discrète dans le temps. Il existe plusieurs façons de gérer le temps en simulation discrète. On dit ainsi que la simulation est :

- **Dirigée par une horloge** : lorsque l'état du modèle est réévalué à intervalles réguliers. Dans la figure 1.2, un intervalle Δt sépare deux transitions d'états.
- **Dirigée par les événements** : lorsque des événements arrivant à des intervalles de temps ir-

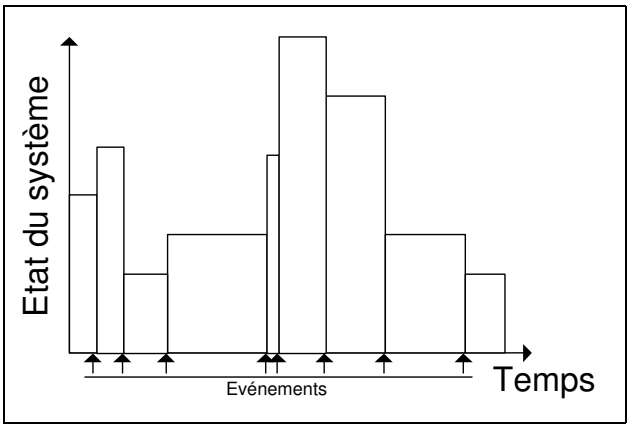

FIG. 1.3*: Simulation à événements discrets.*

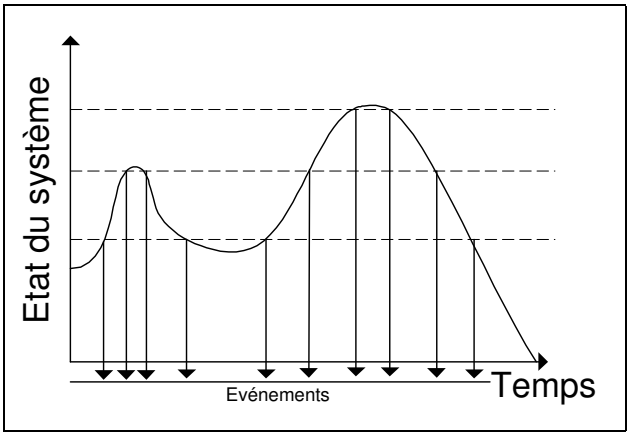

FIG. 1.4*: Quantification de système continu*

réguliers déclenchent les transitions d'états (figure 1.3), la simulation est alors à événements discrets.

Les méthodologies de simulation à événements discrets présentent de nombreux avantages sur la simulation dirigée par horloge. En effet, il est possible de simuler un modèle en temps discret grâce aux événements discrets en programmant des événements d'activation à intervalles réguliers. L'intérêt de l'utilisation de la simulation à événements discrets apparaît lorsque le phénomène simulé utilise des échelles de temps très différentes (de l'ordre de la seconde pour une partie du modèle et de l'année pour une autre) ; dans ce cas, si la simulation est dirigée par une horloge, la règle veut que le pas de temps utilisé soit celui du modèle utilisant la plus petite échelle de temps, même si le sous-modèle n'est actif que pendant une petite partie du temps complet de la

simulation. Les événements discrets permettent de ne pas réévaluer l'état du modèle lorsque ce n'est pas jugé nécessaire. Plusieurs travaux [Kofman et S., 2001], [Zeigler, 1998] portent ainsi sur des méthodes de quantification en états de systèmes continus où la résolution de modèle ne se fait plus à intervalles de temps fixes, mais où des événements sont générés lorsque le système dépasse un seuil (ou quantum) comme nous l'illustrons dans la figure 1.4.

Malgré ces cadres de spécification de modèles, bien définis en théorie des systèmes, il est encore impossible d'obtenir un modèle simulable à partir de spécifications informelles d'un système. Des méthodologies ont donc été développées pour simplifier cette procédure.

1.1.3 Méthodologie d'étude des systèmes

La *modélisation multifacettes* introduite par B.P. Zeigler [Zeigler, 1984] est basée sur les concepts de la théorie des systèmes. Dans cette méthodologie, un modèle est vu comme un moyen de compiler les connaissances sur un système réel. Elle se concentre sur l'organisation de modèles de base pour un domaine. Nous identifierons un domaine (tel l'écologie) comme un ensemble de systèmes qui partagent des attributs, des comportements et des modes de représentations. Pour définir systèmes et modèles en *modélisation multifacettes*, Zeigler définit plusieurs concepts présentés en figure 1.5 :

- **Entité du monde réel** : où l'objet, peut montrer un comportement changeant fortement selon le contexte dans lequel il est étudié ou selon les aspects de son comportement qui sont étudiés.
- **Modèles de base** : représentation hypothétique, abstraite des propriétés de l'objet. En particulier, son comportement qui est valide quel que soit le contexte et qui décrit toutes les facettes de l'objet.
- **Système** : objet bien défini du monde réel, observé sous certaines conditions et considérant seulement certains aspects de son comportement.
- **Cadre expérimental** : décrit un ensemble limité de circonstances sous lesquelles un système (réel ou modèle) doit être observé ou être soumis à expérimentation. Le cadre expérimen-

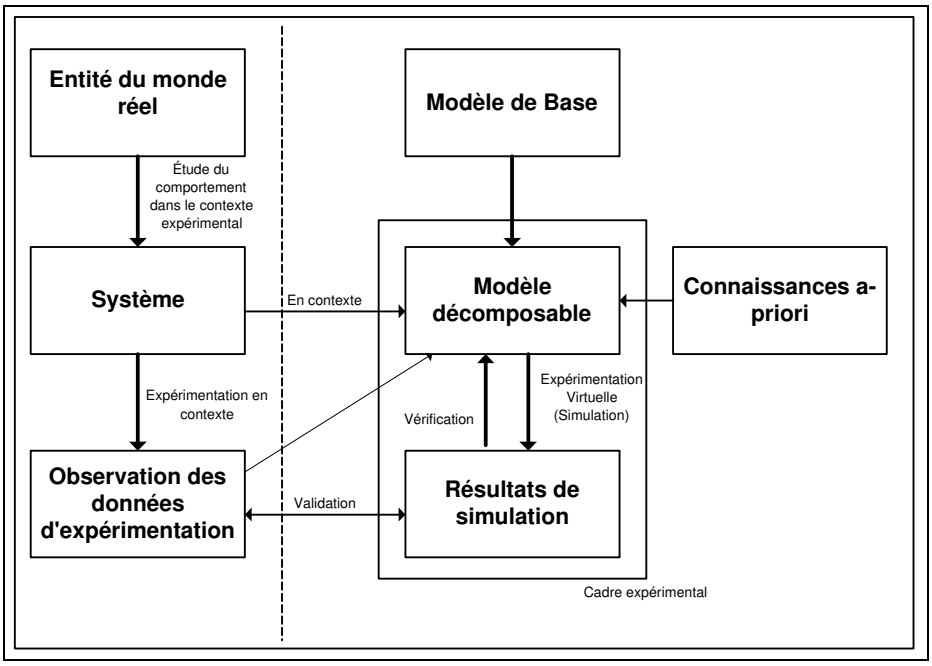

FIG. 1.5*: Concepts de modélisation et simulation [Vangheluwe, 2000]*

tal décrit donc les objectifs de celui qui réalise l'expérience sur le système réel, ou, par la simulation, sur un modèle.

- **Modèle décomposable** : représentation abstraite d'un système dans le contexte d'un cadre expérimental. Ces modèles reflètent généralement certaines propriétés de la structure et/ou du comportement du système (avec un certain degré de précision).
- **Expérimentation** : action de réaliser une expérience. Une expérience peut interférer ou non avec les opérations du système (influencer ses entrées et ses paramètres). En tant que tel, l'environnement d'expérimentation peut être vu comme un système en soi (qui peut lui même être modélisé en modèle décomposable). L'expérimentation induit aussi l'observation et les méthodes de mesure des résultats.
- **Simulation** : simuler un modèle décomposable dans un certain formalisme (tels que les réseaux de Petri [Diaz, 2001], les équations différentielles [Hubbard, 1999] ou les Bond Graphs [Dauphin-Tanguy, 2003]) est le fait de calculer ses dynamiques d'entrées/sorties. La simulation qui permet de "mimer" une expérience réelle peut être vue comme une expérimentation virtuelle. Lorsque le but de la modélisation est de faire la description d'un système

de manière compréhensible et réutilisable, le but de la simulation est d'être rapide et précise. Un point crucial dans la relation système-expérience/modèle-expérimentation virtuelle, est qu'il existe une relation homomorphique entre le système et le modèle : fabriquer un modèle à partir d'un système réel et simuler son comportement devrait alors mener aux mêmes résultats que de faire une expérimentation réelle puis d'observer et codifier les résultats expérimentaux.

- **Vérification** : Processus d'inspection de l'intégrité de résultats d'un programme de simulation en respect du modèle décomposable dont il est dérivé.
- **Validation** : Processus de comparaison entre les mesures et les résultats de simulation dans le contexte d'un cadre d'expérimentation précis. Quand la comparaison montre des différences, le modèle formel qui a été décrit peut ne pas correspondre au système étudié. Néanmoins, un grand nombre de comparaisons avec des résultats de simulation ne valide pas le modèle. Il est important de noter que la correspondance entre le comportement du modèle et celui du système n'est valable que dans le contexte du cadre d'expérimentation. Par voie de conséquence, lorsque des modèles sont utilisés pour échanger des informations, un modèle doit toujours être regardé dans son contexte. De même, un modèle ne doit pas être développé sans que l'on s'intéresse simultanément à son cadre expérimental.

Le point central de la méthodologie de modélisation multifacettes est le mode de représentation **Structure Entité Système** [Zeigler, 1984], qui reconnaît le besoin de représenter :

- **La décomposition** : comment un système peut être décomposé en composants du système.
- **Le couplage** : comment ces composants peuvent recomposer le système original.
- **La taxonomie** : les variantes admissibles d'un composant et ses spécialisations.

Ziegler a aussi proposé la modélisation modulaire et hiérarchique comme un moyen d'exprimer la décomposition, le couplage et la taxonomie. La modélisation modulaire et hiérarchique est une approche de la dynamique des systèmes complexes où des blocs de constructions modulaires (des composants systèmes aux interfaces bien définies) sont couplés pour reformer un système complexe. Ces blocs de construction modulaires sont définis en spécifiant leurs interfaces d'entrées et de sorties sous la forme de ports au travers desquels s'effectuent toutes les interactions.

Les modèles atomiques et couplés seront distingués en modélisation des systèmes. Alors que la structure interne d'un modèle atomique est spécifiée en termes d'états et de transitions d'états, la structure interne d'un modèle couplé est spécifiée par les composants qu'il contient et par leurs interconnexions. Cette modularité forme les bases pour des composant réutilisables car des modèles partageant les mêmes interfaces sont interchangeables. Les concepts de modélisation modulaire et hiérarchique ont été appliqués dans de nombreux domaines, notamment en modélisation de circuits électroniques [Kofman et al., 2000], [Capocchi et al., 2003] et en modélisation physique [Filippi et al., 2001]. C

1.1.4 Modèles modulaires et hiérarchiques et formalisme DEVS

B.P. Zeigler [Zeigler, 1984] proposa le formalisme DEVS (Discrete Event System Specification) pour permettre la formalisation de modèles modulaires et hiérarchiques. Ce formalisme est basé sur la théorie des systèmes ; il permet une représentation formelle de modèles susceptible de manipulations mathématiques comparables aux équations différentielles pour les systèmes continus. Il est possible de procéder à des vérifications formelles de modèles DEVS, ce qui est une aide précieuse lors de la conception du modèle [Freigassner et al., 2000] .

Un modèle atomique DEVS est une structure :

$AM = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, t_a >$ Avec :

- X : $\{(p,v)|$(p∈ ports d'entrée, v∈ Xp)$\}$: Ensemble de ports et de valeurs d'entrée pour la réception d'activations externes,
- Y : $\{(p,v)|$(p∈ ports de sortie, v∈ Yp)$\}$: Ensemble de ports et de valeurs de sortie pour l'émission d'événements de sortie,
- S : L'ensemble des états séquentiels internes,
- $\delta_{int} : S \rightarrow S$ La fonction de transition interne qui place le modèle dans l'état suivant après le temps renvoyé par la fonction d'avancement du temps,
- $\mathbf{t}_a : S \rightarrow \mathbb{R}^+$: La fonction d'avancement du temps qui renvoie le temps de vie de l'état courant (temps jusqu'à la prochaine transition interne),

- δ_{ext} : $Q \times X \rightarrow S$ La fonction de transition externe qui programme les changements d'états en fonction d'activations d'entrées,
- $\lambda : Q \rightarrow Y$: La fonction de sortie qui génère des événements de sortie juste avant la transition interne.

Interprétation :

- Q = $\{(s,e)|(s \in S, 0 < e < ta(s)\}$ est l'ensemble total des états du modèle.
- *e* représente le temps écoulé depuis la dernière transition, et *s* l'ensemble partiel d'états pour la durée de $ta(s)$ en absence d'activation externe.
- δ_{int} : Le modèle étant dans un état *s*, à *ti* il passera dans l'état *s'*, *s'* =δ_{int}*(s)*, en l'absence d'activation externe pendant la durée de $ti + ta(s)$.
- δ_{ext} : Lorsqu'une activation externe se manifeste, le modèle étant dans un état *s* depuis le temps *e* passe en *s'* =δ_{ext}*(s,e,x)*.
- L'état suivant dépend du temps écoulé dans l'état courant.
- A chaque changement d'état *e* est remis à 0.
- λ : La fonction de sortie est exécutée avant la transition interne, avant l'émission d'une sortie le modèle est dans un état passager (transcient).
- Un état avec un temps de vie infini est un état passif, sinon, c'est un état passager (transient). Si un état *s* est passif, le modèle ne peut évoluer qu'avec l'apparition d'activations d'entrée.

Un modèle DEVS couplé CM peut se décrire sous la forme :

$CM = < X, Y, D, \{M_d \in D\}, EIC, EOC, IC >$

- X : Ensemble de ports et de valeurs d'entrée pour la réception d'activations externes,
- Y : Ensemble de ports et de valeurs de sortie pour l'émission d'événements de sortie,
- D : L'ensemble des composants qui lui sont attachés (couplés ou atomiques),
- M_d : Le modèle DEVS pour chaque $d \in D$,
- EIC : L'ensemble des liens d'entrée qui connectent les entrées du modèle couplé à une ou plusieurs des entrées des composants qui lui sont attachés.
- EOC : L'ensemble des liens de sortie qui connectent les sorties d'un ou plusieurs des composants qui lui sont attachés, aux sorties du modèle.

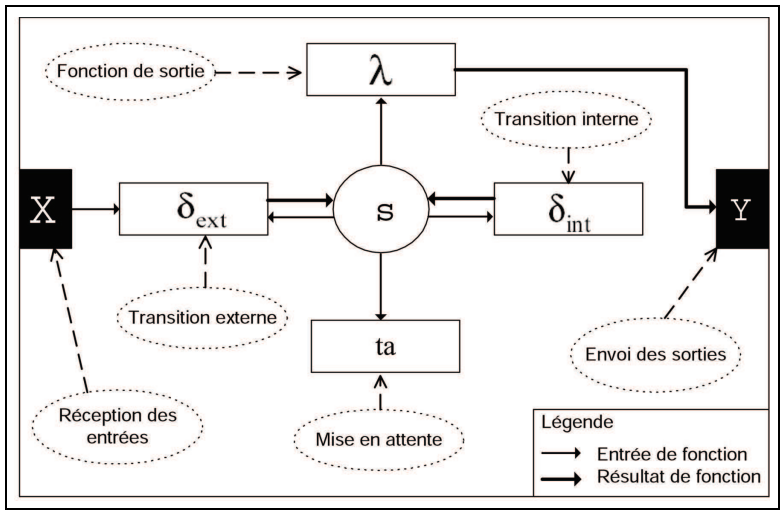

FIG. 1.6*: Fonctionnement d'un modèle atomique*

– *IC* : L'ensemble des liens internes qui connectent les composants qui sont attachés, à *CM* entre eux.

Dans un modèle couplé, un port de sortie d'un modèle $M_d \in D$ peut être connecté à l'entrée d'un autre $M_d \in D$ mais pas directement à lui-même. Modèles couplés comme atomiques sont autonomes et peuvent être stockés séparément. Suivant les implémentations, la structure interne d'un modèle couplé peut être cachée pour créer des composants de plus haut niveau.

Le formalisme DEVS présente une séparation explicite entre la modélisation et la simulation : un modèle DEVS est directement simulable dans le contexte d'un cadre d'expérimentation donné.

La simulation de modèles DEVS est dirigée par les événements qui activent les transitions d'états des modèles. La figure 1.6 présente le fonctionnement d'un modèle atomique en rapport à ces activations.

DEVS est très proche de la théorie générale des systèmes et se présente comme formalisme unificateur pour la spécification de systèmes. Il présente aussi l'avantage d'être *fermé sous composition*. Un formalisme est défini comme *fermé sous composition* si n'importe quelle composition obtenue par couplage de composants spécifiés par le formalisme est elle-même spécifiée par le formalisme. Les équations différentielles et les machines séquentielles sont connues pour être *fermées sous composition*. La signification d'une telle propriété est qu'elle facilite la construction hiérarchique des modèles par l'application récursive des procédures de couplage [Zeigler, 1984].

Récemment [Vangheluwe, 2000] a démontré dans son approche de méta-modélisation, que des modèles conçus dans une grande variété de formalismes (Petri-nets, ODE, State charts, Bond Graphs, etc..) peuvent être représentés avec une grammaire DEVS. Vangheluwe, un des fondateurs de Modelica [MODELICA, 2002], développe l'outil Atom3 [Vangheluwe, 2000] qui permet la transformation automatique de ces formalismes en DEVS au niveau sémantique. La sémantique DEVS est d'ailleurs en cours de standardisation par le *Simulation Interoperability Standards Organization* (SISO) [SISO, 2003].
Néanmoins DEVS doit être adapté et étendu lorsqu'il est replacé dans le contexte spécifique d'un domaine d'application. Le formalisme étant souvent trop abstrait, il est nécessaire d'enrichir sa syntaxe pour permettre de simplifier la spécification de types de modèles particuliers.

Pour cela un grand nombre de techniques dérivées sont l'objet de recherches. Parmi ces techniques certaines se révélant utiles dans le domaine de l'étude de systèmes complexes : F-DEVS [Kofman et al., 2000] pour l'étude de modèles pouvant admettre des comportement ou une structure fautive, DS-DEVS [Barros, 1997] pour la modélisation et la simulation de modèles à structure variable, G-DEVS [Giambiasi et al., 2002] pour l'étude de systèmes hybrides continus/discrets ou encore Fuzzy-DEVS [Kwon et al., 1996] qui permet l'expression de modèles DEVS en utilisant la logique floue. Les formalismes dérivés ont tous été développés pour aider à résoudre des problèmes dans des domaines spécifiques. Certaines de ces techniques, telles Cell-DEVS [Wainer et Giambiasi, 2001] pour la simulation d'automates cellulaires ou JAMES [Schattenberg et Uhrmacher, 2001] pour la modélisation de systèmes multi agents, sont l'adaptation de techniques de modélisation au formalisme DEVS et prouvent aussi qu'une grande variété de techniques peut être adaptée dans ce formalisme. Néanmoins, malgré les possibilités évidentes de couplages entre ces différentes techniques utilisant une sémantique semblable, une seule est généralement disponible dans les environnements de modélisation actuels. Or plus la complexité et la taille du système à modéliser augmente, plus il est nécessaire de mélanger les techniques de modélisation.

Un modèle d'un type spécifique est en général mieux adapté pour la simulation d'une partie du système ou pour une vue spécifique du système. La composition de différents types de modèles

et leurs abstractions à différents niveaux est donc indispensable pour pouvoir étudier un système dans son ensemble. Une telle association de modèles est apellée multi-modèles.

1.2 La multi-modélisation

La simulation d'un modèle est effectuée si une description mathématique suffisante du système ne peut être fournie pour répondre aux questions que l'on se pose d'une manière analytique. Nous avons vu que le formalisme DEVS et sa sémantique étaient adaptés pour définir un environnement de modélisation et de simulation hiérarchique dans lequel les modèles peuvent être facilement agrégés et décomposés tout en garantissant leur intégrité. Aucun modèle n'étant adapté à tous les problèmes, il est évident qu'il est nécessaire de réaliser des compositions pour pouvoir répondre à des questions complexes. Pour cela Ören [Orën, 1991] a développé le concept de Multi-modélisation, plus tard étendu par Fishwick [Fishwick, 1995].

1.2.1 Description des multi-modèles

Du fait qu'un type de modèle ne peut à lui seul simuler qu'une partie de l'ensemble, il faut recourir à un multi-modèles qui est un modèle composé de manière récursive d'autres modèles qui peuvent être de différent types. Nous définissons un type de modèle comme un ensemble de modèles utilisant une même technique de modélisation. La technique de modélisation (automate cellulaire, diagrammes de blocs, multi-agents) est différenciée du formalisme (DEVS, machines à états finis, équations différentielles). La technique décrit une organisation et une conceptualisation particulière pour un modèle tandis qu'un formalisme définit une grammaire et éventuellement un processus de résolution pour décrire ces modèles.

Un multi-modèles présente plusieurs niveaux de description en généralisant ou spécialisant un système et ses parties de plus en plus à chaque niveau. Le plus haut niveau est constitué de "boites noires", (c.à.d des composants dont la fonctionnalité est cachée à leur niveau et révélée à des niveaux plus profonds ; donc plus le niveau est élevé, moins il y a de composants. Le principe régissant de tels modèles est appelé *agrégation et décomposition* [Zeigler, 1984]. Le couplage

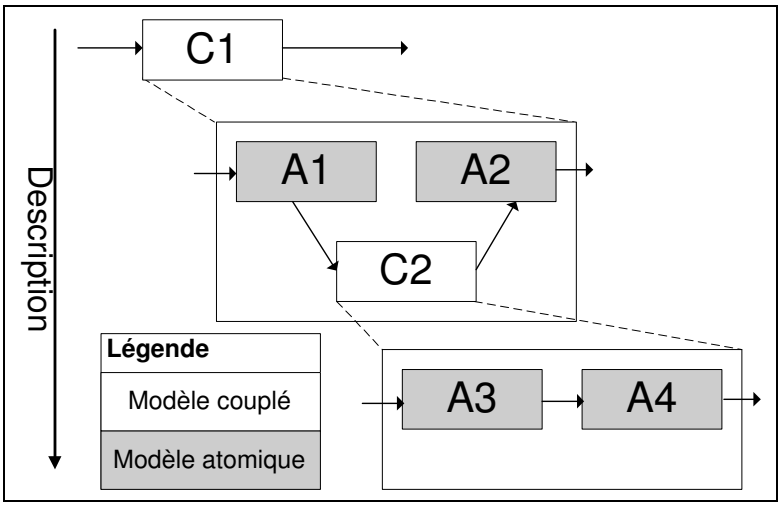

FIG. 1.7*: Décomposition de systèmes hiérarchiques*

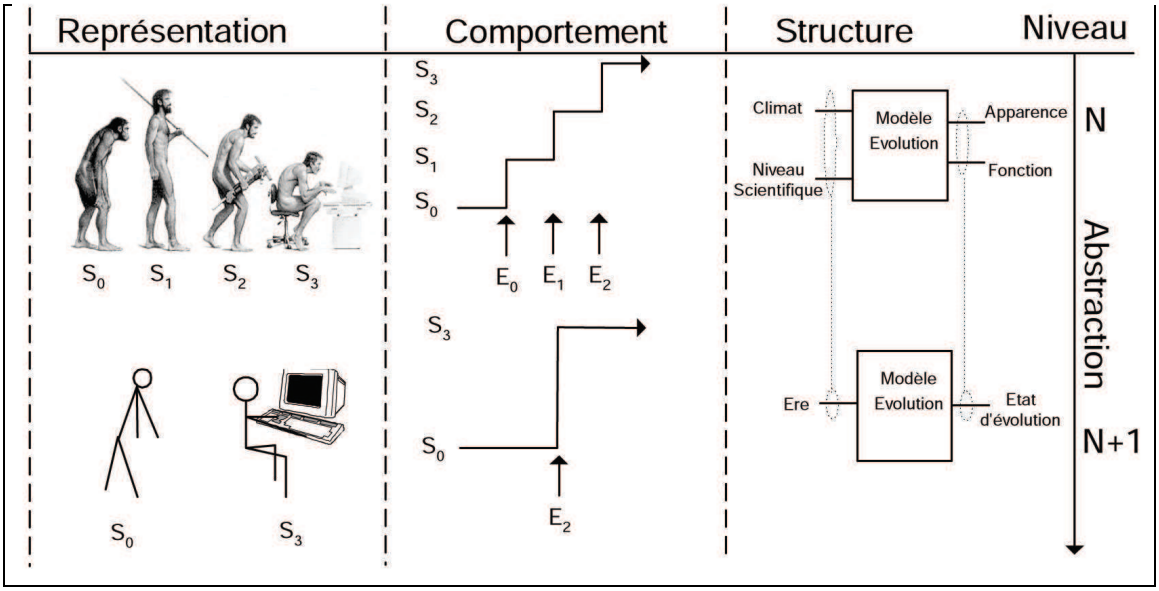

FIG. 1.8*: Exemple d'abstraction d'un certain modèle de l'évolution humaine, du moins abstrait (en haut) au plus abstrait (en bas).* $S_{(0..3)}$ *représente les états du système,* $E_{(0..2)}$ *les événements provoquant les changements d'états*

interne d'un composant est défini au plus bas niveau, mais l'interface des composants est conservée pour permettre le couplage de composants de même niveau. Enfin, les informations fonctionnelles ne sont pas transférées d'un niveau plus bas aux plus hauts. Le niveau le plus bas est, en général, le seul niveau simulable. La figure 1.7 : présente la décomposition hiérarchique du modèle de composition C1 en modèles A1, A2 et C2, et la décomposition du modèle C2 en modèles A3 et A4 : ici seuls les modèles atomiques constituant le niveau le plus bas de la hiérarchie de description sont simulables.

Les multi-modèles présentent aussi plusieurs niveaux d'abstraction en généralisant un système et ses éléments à chaque niveau. Le principe régissant la hiérarchie d'abstraction est appelé *abstraction et spécialisation*. Au plus bas niveau un composant dispose de toutes les propriétés prévues. A de plus hauts niveaux ses propriétés sont réduites et les informations concernant ces propriétés (comportement et états) sont passées à travers les niveaux. Toutefois, quel que soit le niveau, la structure et l'identité des composants restent inchangée. Les interactions entre les composants du modèle sont donc d'autant plus simplifiées que leur niveau d'abstraction est haut. La Figure 1.8 présente un modèle de l'évolution de l'homme à deux niveau d'abstraction. Le niveau d'abstraction le plus bas, N, propose la représentation la plus conforme au monde réel. Le comportement au niveau N est complexe, avec quatre états successifs et trois événements pouvant provoquer les transitions entre états. La structure du modèle au niveau N présente des interfaces d'entrée à deux variables (Climat et Niveau Scientifique) pouvant recevoir les événements externes susceptibles de provoquer des changements d'états. Au niveau d'abstraction plus haut $N+1$ la représentation est simplifiée. A haut niveau, seuls les détails intéressant l'analyse du système dans un certain cadre expérimental sont gardés. Le comportement a lui aussi été simplifié, de quatre états, le modèle n'en contient plus que deux, on parle alors de *compression d'états*. Les événements régissant la transition entre états ont par conséquence été simplifiés, on parle dans ce cas de *compression d'événements*. La quantité d'information contenue dans le modèle au niveau $N+1$ est moindre que dans le niveau N, pourtant la structure du modèle reste identique. En passant du niveau N à $N+1$ les interfaces se sont simplifiées, ayant des entrées composites, par exemple les variables *climats* et *niveau scientifique* s'étant recomposées dans une variable plus générale : *l'ère*.

Hiérarchie d'abstraction et de description sont les concepts de base servant à la création d'environnements de multi-modélisation Reste ensuite la délicate étape du choix du paradigme de programmation permettant d'implémenter ces environnements. Tel qu'il est présenté par Frick [Frick, 2000] et Fishwick [Fishwick, 1995], le paradigme Objet présente des analogies fortes avec les multi-modèles [Oussalah et al., 1995]. Ces analogies ont imposé la programmation objet comme mode d'implémentation d'environnements de multi-modélisation.

1.2.2 Modélisation orientée objet et multimodèles

La programmation Orientée Objet prend également ses origines dans le domaine de la simulation. Le premier langage orienté objet, *Simula* [Dahl et Nygaard, 1966], à été développé à partir d'Algol pour simplifier la simulation de systèmes réels. Tout comme des modèles hiérarchiques, les objets encapsulent les structures de données avec les opérations associées, reçoivent et envoient des messages pour communiquer. Les objets peuvent contenir des références à d'autres objets, créant ainsi des relations similaires aux notions de couplages des multi-modèles. Pour réaliser une action spécifique un objet peut donc faire suivre toute ou partie de cette action à d'autres objets. Les objets peuvent donc être composés d'autres objets formant un tout, concept similaire à la hiérarchie de description.

Les objets ayant des propriétés identiques mais des identités distinctes sont des instances d'une classe. Les propriétés et méthodes d'une classe peuvent être héritées, créant ainsi une hiérarchie d'héritage. Les propriétés pouvant être ajoutées ou modifiées à chaque héritage. Dans ce cas les méthodes appropriées pour réaliser une action spécifique sont cherchées en remontant cette hiérarchie. L'héritage correspond donc à la hiérarchie d'abstraction en multi-modélisation ; les classes parentes sont autant d'abstractions des classes qui en héritent car elles présentent un comportement et des propriétés moins détaillées.

Les concepts d'analyse de modèles orientés objets ont abouti à plusieurs outils d'analyse, parmi lesquels le language UML (Unified Modelling Language) reconnu aujourd'hui comme standard de description de modèles objets. Dans le cadre de la multi-modélisation, Fishwick [Fishwick, 1995] a introduit la méthodologie OOPM (Object Oriented Physical Modelling) réunissant les deux

disciplines pour étendre les analyses classiquement réalisées dans le domaine de la simulation [Hill, 1996]. Cette méthodologie propose une conception objet des modèles pour permettre une réutilisation simplifiée des composants de modélisation. Toutefois, les disciplines de la simulation et de la conception orientée objet restent distinctes et bien des concepts ne sont pas transposés d'une discipline à l'autre malgré des ressemblances fortes.

Parmi ces concepts, les "patrons de conception" [Gamma et al., sley] proposent des éléments d'architecture réutilisables qui peuvent être intégrés lors de la conception de multi-modèles et d'environnements de multi-modélisation. Nombre de ces micro-architectures logicielles sont aujourd'hui standardisées et permettent la réduction de la complexité d'un programme car elles offrent des réponses vérifiées à des micro-problèmes se posant couramment en programmation orientée objet.

Ces travaux ont servi de base à la définition d'un environnement générique de modélisation orienté objets développé dans notre laboratoire [Aiello, 1997], [Delhom et al., 1995], [Delhom, 1997] et [Chiari et al., 2000b]. Ainsi, la multi-modélisation, la conception orientée objet et le formalisme DEVS constituent le fondement de notre travail de création d'un environnement logiciel de multi-modélisation et de spécification des techniques de modélisation leur étant associées.

1.3 Spécifications du problème

L'étude des systèmes physiques pour l'environnement constitue la problématique centrale de la recherche du laboratoire UMR CNRS 6134. Dans ce cadre, nous proposons une contribution à la conception d'un environnement de modélisation et de simulation, ou encore *cadriciel* de modélisation et de simulation permettant l'étude de ces systèmes. Pour permettre la résolution de ces problèmes le *cadriciel* devra aussi être enrichi par des méthodes originales de modélisation et de simulation permettant son utilisation directe au sein de la communauté. Cet environnement devra donc présenter plusieurs propriétés, certaines étant offertes par l'application des concepts de multi-modélisation, d'autres étant le fruit d'un travail original d'intégration de techniques nouvelles :

- Les modèles de simulation doivent êtes faciles à créer et à réutiliser. La modélisation est une activité complexe qui peut être simplifiée par l'utilisation d'une partie d'un modèle déjà existant.
- Le processus de modélisation doit être partagé au sens où, la conception de modèles complexes nécessite l'expertise de plusieurs disciplines. Il est nécessaire d'offrir un environnement donnant la possibilité de documenter, organiser et protéger les modèles, afin d'en simplifier la spécification par des groupes de modélisateurs à travers le réseau.
- L'action commune doit être basée sur la modélisation orientée objet, la multi-modélisation et prendre en compte la hiérarchie d'abstraction et de description, pour simplifier la réutilisation des modèles.
- Une architecture globale modulaire doit être définie pour permettre l'ajout éventuel de nouvelles techniques de modélisation au cadriciel, d'autant que par l'utilisation de patrons de conception il est possible de garder une architecture logicielle permettant une extension future simplifiée. Un grand nombre de techniques informatiques sont déjà appliquées de manière indépendante au domaine de la simulation de systèmes naturels, le lecteur intéressé trouvera dans [Coquillard et Hill, 1997] le détail de ces techniques.
- Le formalisme utilisé doit être suffisamment expressif pour modéliser des systèmes non linéaires, pluridisciplinaires, spatialisés et hybrides continus-discrets. DEVS présente ces propriétés, mais il reste toutefois trop abstrait pour être utilisable par un modélisateur qui ne serait pas spécialiste. Des techniques de modélisation originales devront donc être proposées afin de simplifier la spécification de modèles.
- Le cadriciel devra offrir des possibilités de prise en compte de phénomènes spatiaux car une grande partie des problèmes rencontrés sont des phénomènes ayant une dimension spatiale. Des techniques de modélisation originales doivent donc être intégrées à l'environnement, tout comme la possibilité d'interconnexion avec des SIG (systèmes d'information géographiques).
- Une interface graphique devra faciliter la spécification, la maintenance et la publication des modèles. Les différents types de données à traiter, tout comme les différents utilisateurs des

modèles, créent des modes d'interaction et de présentation spécifiques auxquels devront être adaptés une série de modules interconnectés au noyau de modélisation et de simulation.

1.4 Conclusion

L'examen liminaire des concepts régissant la description et l'utilisation de modèles de systèmes physiques complexes et de la nécessité du recours à la multi-modélisation, méthode nécessaire pour permettre la description de systèmes complexes, nous invitent à opter pour le formalisme DEVS dont nous avons montré la pertinence comme base unificatrice permettant la coopération de différents types de modèles dans un environnement de multi-modélisation.

Afin de positionner plus clairement notre travail, nous nous devons de présenter dans le chapitre qui suit, une revue représentative des approches logicielles de modélisation de systèmes complexes existants.

CHAPITRE 2

L'existant

Ce chapitre aborde les principales approches logicielles actuelles dans le domaine de la modélisation et de le simulation de systèmes naturels complexes. Si nous nous limiterons à présenter une revue partielle et partiale des environnements de modélisation disponibles c'est parce que la plupart de ces approches s'applique à des domaines trop spécifiques qui n'ont pas leur place dans notre démarche. Nous omettrons par exemple, les logiciels commerciaux tels qu'Arena [Arena, 2003], Stella [Stella, 2003], Berkeley Madonna [Madonna, 2003] ou mathlab/simulink [Mathworks, 2003] qui abordent le problème de la modélisation et de la simulation de systèmes naturels complexes, dans une approche trop différente de celle qui nous préoccupe. En somme, notre choix répond au souci d'éviter un inventaire exhaustif pour ne présenter que les approches logicielles s'intégrant efficacement à notre problématique.

C'est pourquoi, dans la premiere partie de ce chapitre, nous évoquerons essentiellement les environnements de multi-modélisation basés sur DEVS et que dans la deuxième, nous ferons une revue sélective des différentes approches logicielles de modélisation de systèmes naturels. Nous serons ainsi en mesure, à partir d'une synthèse de leurs propriétés, de définir dans notre dernière

partie, ce que l'existant identifie comme caractéristiques d'un logiciel s'inscrivant dans notre problématique.

2.1 Environnements de multi-modélisation basés sur DEVS

Cette section présente une revue des environnements basés sur DEVS s'inscrivant au plus près de notre problématique. Le lecteur interessé trouvera une liste commentée des outils basés sur DEVS à l'adresse suivante, [Wainer, 2003], (actualisée par G.Wainer).

2.1.1 DEVSJAVA/Collaborative Devs Modeller

DEVSJAVA [Sarjoughian et Zeigler, 1998], développé à l'Université de l'Arizona, est un environnement écrit en Java, basé sur DEVS et le paradigme objet. Il supporte la norme "High Level Architechture" (HLA) [Zeigler et al., 1999], la modélisation basée sur les agents et la Structure Entité Système présenté dans le chapitre précédent. L'environnement DEVSJAVA fournit les classes de base permettant d'étendre la méthodologie DEVS en utilisant les capacités du langage Java. Cet environnement est couplé à l'interface CDM (Collaborative DEVS Modeler) qui permet à des modélisateurs de composer ensembles des modèles modulaires et hiérarchiques bien que séparés géographiquement. [Sarjoughian et Zeigler, 1999]. Des groupes d'experts et de professionnels utilisant la modélisation peuvent collaborer avec cet outil de manière synchrone ou asynchrone à travers des diagrammes structurés partagés. CDM oblige à respecter les règles de composition nécessaires pour assurer que les modèles créés sont conformes avec le paradigme de modélisation DEVS.

2.1.2 Moose

MOOSE [Fishwick, 1998] (Multimodel Object Oriented Simulation Environment) est une architecture d'application pour la modélisation et la simulation développée à l'université de Floride.

Cet environnement a été développé pour expérimenter les avancées réalisées dans ce laboratoire par Paul Fishwick et son équipe en multi-modélisation [Fishwick, 1995].

MOOSE est basé sur OOPM (Object Oriented Physical Modeling), technique logicielle qui étend les concepts généraux de la programmation orientée objets en précisant les rapports entre modèles et objets. OOPM fournit des mécanismes naturels pour la modélisation de systèmes utilisant de grandes échelles d'espace et facilite l'intégration de multiples partitions de codes dans une simulation.

MOOSE est composé d'une interface utilisateur, d'une bibliothèque de modèles et d'un moteur de modélisation et de simulation. L'interface graphique permet au modélisateur de composer le modèle, de contrôler son exécution et de visualiser les résultats. La bibliothèque se définit alors comme un lieu de stockage des modèles, lieu qui facilite leur définition dans une action partagée ainsi que leur pérennité. Le moteur de modélisation et de simulation permet, pour sa part, de convertir automatiquement une définition de modèle en un programme de simulation exécutable pouvant être contrôlé par l'interface graphique.

Les types de modèles dynamiques pouvant être intégrés à MOOSE sont les modèles en diagrammes de blocs, les modèles à équation sous contraintes et les modèles à base de règles. Un utilisateur peut aussi ajouter des méthodes spécifiques en développant un code propre en C++. Les modèles de différents types peuvent ensuite être combinés grâce à la multi-modélisation qui permet de connecter ces different types de modèles de manière récursive et hiérarchique. La représentation permettant la multi-modélisation dans MOOSE est appelée "Block". Block est un langage propre à définir des composants présentant des interfaces d'entrée/sortie, un peu à la manière des langages de création de circuits électroniques comme VHDL [Aumiaux, 2000].

Les derniers développements de MOOSE concernaient l'intégration de DEVS comme formalisme unificateur du logiciel [Fishwick, 1998]. Toutefois son développement est actuellement arrêté. L'expérience acquise pour la création de cet environnement à été capitalisée par leurs auteurs en vue du développement de son successeur *Rube* [Hopkins et Fishwick, 2003]. *Rube* possède les mêmes caractéristiques que MOOSE mais utilise une interface utilisateur en trois-dimensions (3D)

pour la création de modèles. Les modèles créés par *Rube* possèdent eux mêmes une représentation 3D et utilisent les format XML et X3D [Web3D, 2003] comme moyen de stockage et d'affichage.

2.1.3 Atom3

AToM3 (A Tool for Multi-formalism and Meta-Modelling) [Vangheluwe et al., 2002] est un outil de modélisation multi-paradigmes développé par le laboratoire MSDL (Modelling, Simulation and Design Lab) à l'université de McGill Montréal, Canada. Cet outil a été conçu en collaboration avec Juan de Lara de Universidad Autónoma de Madrid (UAM), Espagne.

Les deux principales fonctionnalités d'AToM3 sont la méta-modélisation et la transformation de modèles. La méta-modélisation est la description ou la modélisation de différents types de formalismes. La transformation de modèles est une technique consistant à transformer, traduire ou modifier automatiquement un modèle décrit dans un certain formalisme, vers un autre modèle qui peut être décrit soit dans le même formalisme soit dans un autre.

Dans AToM3 les formalismes et les modèles sont décrits comme des graphes. Cet environnement génère un outil de manipulation graphique des modèles décrits dans un formalisme donné, à partir d'une méta-spécification de ce formalisme (décrite dans le formalisme entité/relation). Les transformations entre modèles sont ensuite effectuées par réécriture des graphes et peuvent être décrites comme ces derniers.

Toutefois il n'est pas toujours possible de réaliser la traduction de modèle d'un formalisme vers un autre sans pertes d'informations. L'approche adoptée par Vangheluwe dans [Vangheluwe, 2000] pour résoudre ce problème consiste alors en une transformation de tous les modèles sous forme DEVS. DEVS est ainsi utilisé comme formalisme unificateur à partir duquel tous les modèles peuvent être soit simulés soit transformés.

2.1.4 Cell-DEVS

Cell-DEVS est un environnement dédié à la modélisation de systèmes pouvant être représentés sous la forme d'espaces cellulaires actifs.

Cell-DEVS se base sur le formalisme des automates cellulaires [Ilachinski, 2001] qui est très utilisé pour décrire de tels types de modèles. Les automates cellulaires évoluent en exécutant une fonction de transition globale qui met à jour l'état de toutes les cellules.

Le comportement de cette fonction globale dépend du résultat d'une fonction s'exécutant localement dans chacune des cellules, de manière synchrone et parallèle en utilisant l'état des cellules voisines. La simulation de tels modèles s'effectuent donc en temps discret ce qui en réduit la précision et l'efficacité. Il est en effet peu courant d'avoir à actualiser toutes les cellules en même temps.

Cell-DEVS, tout en étant conforme au formalisme de base par le respect de la simulation à événements discrets et la modélisation basée sur DEVS , propose de résoudre ces problèmes. La principale différence entre cet environnement et le paradigme des automates cellulaires se situe dans la définition du comportement des cellules qui est ici une fonction dépendante du temps de vie de l'état des cellules. Les extensions proposées offrent un mécanisme simple pour définir des délais explicites pour chaque cellule. L'outil résultant de cette méthode permet la modélisation et la simulation de domaines spatiaux composés de cellules. La modélisation d'un domaine s'effectue par la description de la dynamique d'une cellule dans une syntaxe très simplifiée au regard d'un langage classique.

A l'issue de ce tour d'horizon des outils permettant la modélisation et la multi-modélisation à l'aide de DEVS, il semble important d'examiner quelques logiciels disponibles dans le domaine qui nous intéresse, la modélisation de systèmes naturels.

2.2 Environnements de modélisation de systèmes naturels

Cette section présente quatre logiciels utilisés pour la modélisation de systèmes environnementaux. Notre choix s'est volontairement limité à ces approches car elles ont été développées exclusivement pour la résolution de problèmes liés à ce domaine. Ce choix constitue donc une base représentative des tendances actuelles dans le domaine précis des logiciels de modélisation de systèmes naturels.

2.2.1 SWARM

Swarm [SWARM, 2002] est un ensemble de bibliothèques qui facilitent l'implémentation de modèles basés sur les agents. L'inspiration de Swarm trouve sa source dans le domaine de la vie artificielle. La vie artificielle constitue en effet, une approche d'étude de systèmes biologiques qui essaye de reproduire de manière artificielle des phénomènes, par identification de dynamiques communes à un ensemble de systèmes biologiques.

Deux types de stratégies utilisant la vie artificielle se sont ainsi avérées utiles dans de nombreux domaines. Une stratégie d'évaluation empirique de la dynamique : la combinaison d'entités autonomes dans un environnement partagé étant un processus récursif non solvable analytiquement. La seconde est la synthèse de connaissances. La vie artificielle cherche à extrapoler les connaissances biologiques pour suggérer de nouvelles expérimentations. Dans les deux cas Swarm peut être utilisé comme moyen d'expérimentation logicielle.

Swarm, conçu dans le souci de réutilisation potentielle de modèles, offre au modélisateur désirant étudier un système, une bibliothèque de composants. Le développement d'un modèle se fait à travers la définition de classes composant les entités du modèle. La simulation de ces modèles se faisant ensuite grâce à un ensemble d'interfaces graphiques standards ou redéfinies par le modélisateur en Langage C.

La première version de SWARM, disponible en 1996, à évolué pour être utilisée en biologie ainsi qu'en anthropologie, informatique, écologie, économie, géographie, industrie et sciences politiques.

2.2.2 Tarsier

L'environnement de modélisation Tarsier [Rahman et al., 2001] est un ensemble de logiciels qui permettent le développement et le déploiement rapide d'outils de gestion environnementale. Parmi ces outils nous pouvons dénombrer un module de simulation, un module de stockage, un module d'analyse de données et un module de visualisation. Tarsier peut traiter de nombreux formats de données, raster (grilles), séries temporelles, réseaux et liste de points géographiques.

Sur les données sont construits un ensemble d'outils d'analyses pour l'interpolation, le traitement statistique, l'échantillonnage ou la transformation de données.

Enfin les modules de simulation utilisent les modules d'analyses comme sources de données. Différents types de modéles allant des automates cellulaires aux modèles hydrologiques de bassins versants de type réseaux, peuvent être développés avec cet environnement. Toutefois ce développement de modèles se fait sous la forme de classes C++ très fortement dépendantes des données et donc très peu réutilisables.

Tarsier peut être vu en ce sens comme un SIG dynamique permettant à un utilisateur de composer son système d'analyse propre pour une étude donnée.

2.2.3 ECLPSS

Eclpss (Ecological Component Library for Parallel Spatial Simulation) [Woodbury et al., 2002] est un environnement basé sur le langage Java destiné aux écologues. Cet outil permet d'élaborer facilement des simulations d'écosystèmes dans de nombreuses échelles de temps et d'espace. Un modèle Eclpss est composé de trois entités : un ensemble de variables, un ensemble de composants qui modifient ces états et un ensemble de simulateurs pour ces composants.

Les composants Eclpss sont développés en code java et stockés dans une bibliothèque pour réutilisation. L'intérêt principal de cet environnement est sa capacité à effectuer des simulations sur des machines parallèles.

2.2.4 VLE

VLE (Virtual Laboratory Environment) [Duboz et al., 2003] est un environnement multi-agents et orienté objets pour la modélisation de systèmes complexes développé au LIL (Laboratoire d'Informatique du Littoral). VLE permet la modélisation de systèmes suivant le paradigme multi-agents. L'utilisateur de VLS dispose d'une bibliothèque de fonctions réutilisables pour décrire la dynamique d'un modèle. L'étude d'un système s'effectue par une analyse comparable à une analyse objet avec UML pour décomposer le système en composants. La dynamique de chaque

composant peut ensuite être décrite en utilisant soit des réseaux de Petri interprétés, soit en recourant à un langage algorithmique, soit encore par développement direct en C++, soit enfin au moyen d'un système à base de règles.

Le moteur de simulation de VLE est écrit en C++ ce qui lui confère une certaine rapidité d'exécution. L'interface de visualisation d'expérimentation est écrite en Java par souci de portabilité.

2.2.5 Cormas

CORMAS [Bousquet et al., 1998] est un environnement développé au CIRAD, pour la programmation de modèles multi-agents. CORMAS, spécialisé dans la gestion de ressources renouvelables, offre un cadre de développement de modèles de simulation, des modes de coordination entre des individus et des groupes qui exploitent ces ressources. Le cadre de CORMAS se structure en trois modules :

- un module permettant de définir les entités du système (agents) et leurs interactions,
- un module permettant de contrôler la dynamique de la totalité du modèle,
- un module permettant d'observer la simulation depuis différents points de vue (affichages de différentes propriétés du modèle en cours de simulation).

La plupart des éléments composants ces trois modules sont prédéfinis et disponibles dans des classes SmallTalk [Clavel et al., 1997]. L'interface de développement de CORMAS est l'environnement de développement intégré de VisualWorks[Cincom, 2003].

L'ensemble d'outils ci-dessus présentés est destiné avant tout à la modélisation de systèmes naturels. Pour progresser dans l'identification des caractéristiques nécessaires à la définition de notre approche logicielle, nous devons maintenant opérer une synthèse de leurs principales propriétés.

2.3 Synthèse des différentes approches

Une tendance très présente dans le domaine de la modélisation de systèmes naturels consiste à les simplifier pour pouvoir créer des modèles basés sur les agents. Cette approche est proposée par les environnements CORMAS, VLE et SWARM mais aussi dans des environnements non

présentés par soucis de synthèse, comme JAMES [Schattenberg et Uhrmacher, 2001], basés sur DEVS. Les systèmes à agents sont discrets dans la plupart des dimensions comme le temps et les états. En général, ce type de modèles n'est pas utilisé pour l'obtention de résultats quantitatifs mais plutôt pour vérifier des hypothèses d'évolution ou de dynamique d'un système. Néanmoins il est difficile d'évaluer l'impact des simplifications faites lors de la modélisation, ce qui constitue un inconvénient par rapport à des approches plus classiques sous forme d'automates cellulaires comme celles offertes par Cell-DEVS ou Eclpss.

Nous pouvons aussi noter l'émergence de l'utilisation de normes et standards dans le domaine des objets-modèles communiquants. Ces normes interviennent à plusieurs niveaux allant du langage de programmation (Java pour DEVSJAVA, Eclpss ou VLE), aux formalismes utilisés (notamment DEVS) ou encore des protocoles de simulation comme HLA. Néanmoins cela ne garantit pas l'interopérabilté des modèles au niveau sémantique ce qui se révèle comme un passage obligé si on veut aboutir à une réelle compatibilité entre les différents environnements de modélisation. La définition de grammaire est une problématique au coeur de l'environnement Atom3. Même si dans le cas de ce logiciel l'objectif poursuivi n'est pas directement la standardisation, les auteurs de cet outil affirment que DEVS peut constituer un formalisme unificateur. Nous pouvons noter que dans le cas de modèles basés sur DEVS cet effort est poursuivi par le DEVS Standardisation Group [DEVS-Group, 2003].

Il existe aussi un intérêt croissant pour une intégration plus poussée entre les outils de simulation et les outils de traitement de données comme par exemple les SIG. Un outil tel que Tarsier propose même une intégration totale avec un SIG. La possibilité de stocker, manipuler et analyser différentes couches de données de différentes résolutions correspond tout à fait aux niveaux d'abstractions de modèles et de leurs simulations. Les SIG n'intègrent malheureusement que de façon très limitée la dimension temporelle des données, ce qui pose problème pour l'exploitation de résultats de simulation. Même si un travail important a été réalisé [Hill et al., 2000] des efforts sont encore nécessaires dans ce domaine.

Plusieurs approches logicielles ont fait aussi le choix de la séparation en plusieurs modules exécutant des tâches spécifiques. Ainsi DEVSJAVA/CDM, MOOSE, VLE ou encore Eclpss, COR-

MAS et Tarsier proposent une séparation explicite du moteur de simulation ce qui permet de simplifier son développement tout en assurant l'intégrité générale du logiciel. Eclpss, Tarsier, CORMAS et dans une certaine mesure SWARM et VLE proposent aussi des modules de contrôle de simulation, permettant même d'interagir avec des modèles en cours de simulatio. Cette capacité d'interaction présente un grand intérêt notamment pour le calibrage et la vérification de modèles en vue de la mise en place de scénarios de simulation spécifiques. En ce qui concerne la tâche de modélisation, plusieurs options ont été prises par les différentes approches, sous forme de code avec SWARM, CORMAS, Eclpss, Tarsier et MOOSE, et ce, à l'aide d'un code à syntaxe simplifiée pour VLE et Cell-DEVS ou encore par une approche hybride, mêlant code pour la définition des comportements et interface graphique pour la définition de structure dans DEVSJAVA/CDM.

Plusieurs de ces approches proposent des bibliothèques de modèles. Ainsi MOOSE, SWARM, Eclpss, Atom3 et dans une certaine mesure DEVSJAVA, Cell-DEVS, CORMAS et VLE intègrent des mécanismes pour stocker et récupérer des modèles déjà développés favorisant ainsi la réutilisabilité de modèles. Cette problématique est d'ailleurs très présente dans d'autres approches, non discutées ici, comme celle de [Rizzoli et al., 1998]. Parmi ces environnements MOOSE, VLE et DEVSJAVA permettent même le stockage de modèles différents en nature car ils utilisent des techniques de modélisation différentes. Cependant seul Atom3 propose de manière claire une syntaxe pour le stockage ; cette syntaxe se présente toutefois sous la forme d'une grammaire de graphes et n'est pas directement utilisable dans le cadre d'un logiciel de modélisation. Des outils comme MOOSE et particulièrement DEVSJAVA/CDM mettent le concept de bibliothèque de modèle au centre de leur solution de modélisation partagée. Le concept de bibliothèque est alors étendu pour prendre en compte différents acteurs pouvant intervenir sur ce qu'il serait plutôt convenu d'appeler une base de donnée de modèles.

Enfin, Tarsier, CORMAS ou même Atom3 proposent une approche particulièrement intéressante pour l'exploitation de modèles. Ces logiciels permettent la génération d'interfaces spécifiques selon la technique, le paradigme ou même le modèle à étudier. Nous pouvons imaginer que grâce aux capacités actuelles de publication d'application, notamment le Web, il est tout à fait possible de diffuser largement ces types d'applications. Différents cadres d'expérimentations peuvent

ainsi être générés pour différents utilisateurs finaux du modèle publié. Un travail important reste cependant à effectuer sur la formalisation de ces cadres expérimentaux, en particulier en ce qui concerne les niveaux de détails et les modes d'interactions à associer aux niveaux d'abstraction utilisés dans ces cadres.

Les différentes approches proposées par ces logiciels se révèlent tout à fait complémentaires. La plupart proposent des réponses originales à des problèmes réels soit dans le cadre de la multi-modélisation, soit dans le cadre de l'étude de systèmes environnementaux. Ceci se révèle d'un grand intérêt pour notre problématique et nous invite à présent, à identifier les solutions qui devront être évaluées lors de la conception de notre approche.

2.4 Conclusion

Comme nous l'avons constaté, les logiciels proposent notamment l'usage d'un formalisme unificateur, DEVS, pour faciliter l'intégration de modèles et leurs réutilisabilité. Ce souci de réutilisation est traité de manière moins formelle dans les approches présentées dans la seconde partie de ce chapitre qui traite des environnements de modélisation de systèmes naturels complexes. Cependant, nous devons admettre que ces logiciels proposent des solutions intéressantes aux problèmes spécifiques se posant dans ce domaine. La synthèse proposée dans la troisième partie de ce chapitre nous conduit dès lors à identifier six caractéristiques essentielles qui devront être évaluées lors de la définition de notre approche :

- Plusieurs approches existantes reposent sur l'utilisation d'une modélisation basée sur les agents. Nous ne disposons pas d'assez de recul sur la méthode pour réaliser le choix de l'utiliser comme paradigme fondateur. Il est préférable alors d'opter pour la définition d'un environnement ouvert pour nous réserver la possibilité d'une modification ultérieure des contraintes de cette méthode.
- Il est nécessaire d'utiliser un formalisme unificateur pour permettre l'intégration de modèles hétérogènes tout en assurant l'intégrité du modèle final.

- L'outil de simulation doit offrir des interfaces et des méthodes d'accès aux outils de traitement de données, notamment aux SIG.
- L'architecture proposée doit être modulaire, séparant moteur de modélisation et de simulation, stockage des modèles, cadres expérimentaux et interface graphique de modélisation.
- L'utilisation de bibliothèques de modèles se révèle nécessaire pour assurer la collaboration entre modélisateurs et constitue un élément essentiel à la réutilisation de modèles.
- Les modalités d'expérimentation peuvent être multiples pour un même modèle car il est nécessaire d'offrir des cadres expérimentaux correspondant à l'usage qu'en fera son utilisateur.

Ces caractéristiques répondent en partie aux problèmes posés dans le premier chapitre de cette thèse. Ils constituent ainsi la base de notre cahier des charges pour l'architecture logicielle proposée dans le chapitre suivant.

CHAPITRE 3

Environnement de modélisation

Ce chapitre présente l'architecture générique orientée objet pour un environnement de multi-modélisation et de simulation. Une telle architecture est appelée cadriciel (framework) et présente une organisation de classes permettant d'appréhender un problème spécifique. Le but est identique à celui du concept des patrons de conceptions (design patterns)[Gamma et al., sley], mais un cadriciel est plus spécifique et ne s'applique qu'à un domaine particulier. Plusieurs patrons de conception sont d'ailleurs utilisés dans le cadriciel pour résoudre des sous-problèmes. Il est difficile de définir précisément le terme cadriciel et plusieurs définitions sont admises (e.g. [Johnson et Foote, 1988]) toutefois nous retenons celle de [Campos, 2000] qui semble plus adaptée à notre problématique : *"un cadriciel est une collection d'éléments de conception (e.g. les patrons de conception) et d'implémentation (e.g. les composants logiciels) en coopération et réutilisables qui permettent de créer des applications ou des parties d'applications dans un domaine spécifique"*. Nous proposons dans le cadre de cette étude une collection d'éléments de conception et d'implémentation permettant de résoudre des problèmes dans le domaine de la multi-modélisation.

Le cadriciel est détaillé dans ce chapitre, décomposé en deux parties, la première présente le processus de son développement et la seconde son architecture.

3.1 Développement du cadriciel

La conception d'un cadriciel n'est pas chose facile lorsqu'on veut offrir la généricité à partir de spécifications faites par l'homme. Il est en effet fréquent qu'une architecture et son logiciel dérivé posent des problèmes pendant la phase de test ou lorsqu'ils sont exploités par un l'utilisateur final. Pour éviter de tomber dans ces travers nous avons adopter un cycle de développement différent en associant deux ressources disponibles : la communauté scientifique et la communauté du logiciel libre.

Pour cela, nous avons produit une implémentation du cadriciel dès les premiers instants de son développement. L'environnement de modélisation ainsi créé, JDEVS (Java Discrete EVent Simulator), est présenté dans le chapitre 5 de ce document.

Le cadriciel a ensuite été exposé à la communauté scientifique, par des publications, [Filippi et al., 2002b], [Filippi et al., 2002a] et [Filippi et Bisambiglia, 2003] puis experimenté dans le cadre de contrats de recherches où se sont exprimés les besoins de spécialistes dans des domaines divers, allant de l'écologie à l'énergétique. Les retours d'informations ont permis de mieux définir les rôles de chacun dans le cadriciel ainsi que de nouveaux modes d'interactions ; plus précisément, le travail dans le cadre de contrats de recherches nous a amené à intégrer plusieurs techniques au cadriciel pour répondre à des besoins spécifiques en modélisation. A chaque problème spécifique nous avons apporté une réponse générique en développant des composants spécifiques et en redéfinissant l'architecture générale du logiciel pour qu'il permettre l'intégration de ses composants par héritage. Ces passages successifs du développement spécifique vers le générique nous ont permis d'assurer une architecture modulaire mais maintenant figée à l'environnement.

La seconde ressource utilisée pour contribuer aux spécifications du cadriciel est la communauté du logiciel libre. Pour y accéder nous avons publié le logiciel sur un site de travail partagé,

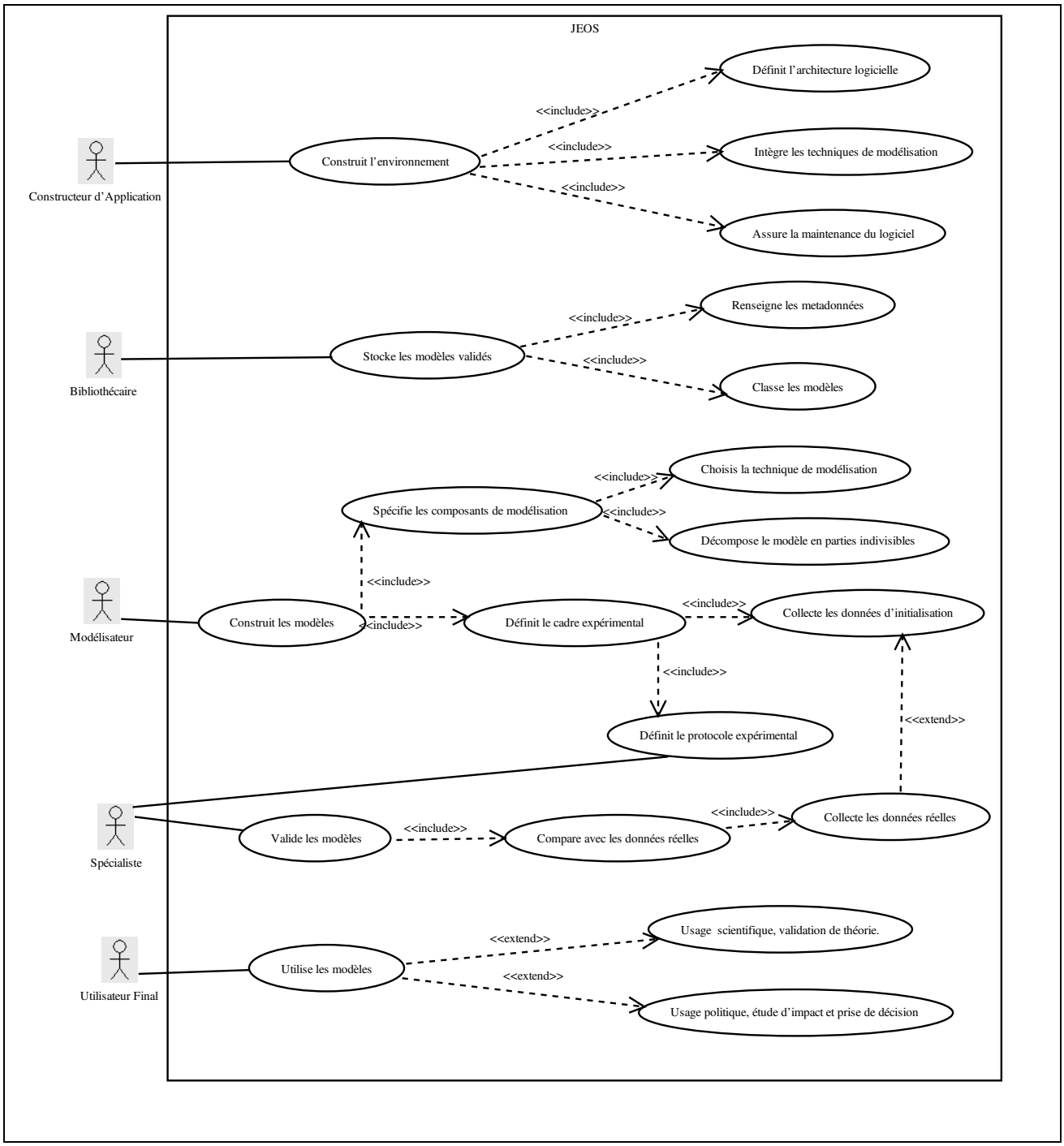

FIG. 3.1*: Cas d'utilisation du logiciel*

Freshmeat [Filippi, 2003] dès qu'une version stable fut disponible. Nous avons pu ainsi inventorier et sérier les problèmes et conseils évoqués par utilisateurs. Ces retours continuels d'informations et la relative hétérogénéité des problèmes à traiter, ont permis d'optimiser la spécification du cadriciel. L'étude des cas d'utilisation redéfinis tout au long du processus de spécification a permis son organisation sous forme de packages. La notation UML [Muller et Gaertner, 1970] est utilisée dans la suite de ce document pour illustrer les différentes parties de notre approche.

La figure 3.1 présente le diagramme final des cas d'utilisation. Nous avons cinq acteurs :

- Le constructeur d'application : il assure la construction du logiciel, son support et de sa mise à jour. Il est aussi chargé d'offrir une réponse logicielle aux besoins d'ajout de techniques de modélisation dans l'environnement.
- Le bibliothécaire : il doit assurer l'entretien de la bibliothèque de modèles c'est-à-dire leur indexation en renseignant des méta-données facilitant leur recherche et enfin leur classement dans une architecture de bibliothèque hiérarchique telle que définie dans [Bernardi et Santucci, 2002].
- Le spécialiste du domaine d'application : il est chargé de fournir les spécifications du modèle à construire, de mettre en place le protocole expérimental permettant de récolter les données réelles et les données d'initialisation. Lorsque cette tâche est partagée, le spécialiste travaille en étroite collaboration avec le modélisateur pour l'implémentation et la validation des modèles.
- Le modélisateur : il est chargé d'implémenter le modèle ; à ce titre il a besoin de connaître parfaitement l'environnement et d'avoir des connaissances basiques en informatique. Il est aussi chargé de choisir la technique de modélisation la plus appropriée et de formater les résultats pour permettre à l'utilisateur final de les analyser.
- L'utilisateur final : il utilise les modèles validés, formate les données d'initialisation et analyse les résultats. Nous avons déterminé deux cas principaux d'utilisation pouvant nécessiter deux types de cadres expérimentaux différents :
  - l'étude de théories nouvelles pour observer virtuellement des systèmes ne pouvant pas être directement étudiés en conditions réelles.

– la prise de décisions, lorsque la simulation est nécessaire pour construire un jugement dans le domaine de l'aménagement du territoire, de la gestion de ressources ou l'élaboration de politiques environnementale.

L'architecture générale du logiciel a été définie à partir de l'analyse de ces cas d'utilisation. Ainsi, les cadres expérimentaux ont été séparés de l'interface graphique, du moteur de stockage et du moteur DEVS, pour faciliter les mises à jour du logiciel selon les requêtes formulées par les différents acteurs. L'environnement est conçu pour permettre l'ajout de nouvelles techniques de modélisation à travers l'intégration de modules externes. La section suivante présente l'architecture générale du cadriciel et les détails de ses différentes parties.

3.2 Architecture du cadriciel

Le cadriciel est composé de quatre packages principaux constituant la partie générique, indépendante des techniques de modélisation utilisées qui seront spécifiées dans des packages différents. La figure 3.2 présente ces quatre packages, le package *moteur-DEVS*, le package *cadres-expérimentaux*, le package *interface-graphique* et le package *stockage*. Les techniques de modélisation (représentées dans la figure 3.2 par le package *technique-de-modélisation*) sont intégrées séparément pour permettre de garder l'environnement ouvert sans avoir à modifier son architecture. Les packages spécifiques aux techniques devront donc contenir un certain nombre de classes héritant des packages principaux, assurant ainsi l'interopérabilité des composants à haut niveau de description. L'accès aux packages des techniques de modélisation se fait grâce à la classe **Access** qui offre les liens vers les classes spécifiques de chaque package. Ceci correspond à une application du patron de conception ***Façade*** [Gamma et al., sley] où un objet **Access** représentant la ***Façade*** pour le sous-système correspond à la technique de modélisation.

Sans ajout de technique spécifique, le cadriciel présente toutefois les classes nécessaires à la définition de modèles DEVS classiques, ce qui représente l'équivalent d'un environnement comme DEVSJAVA/CDM [Sarjoughian et Zeigler, 1998].

Nous présentons par la suite le détail des packages contenus dans la partie générique du cadri-

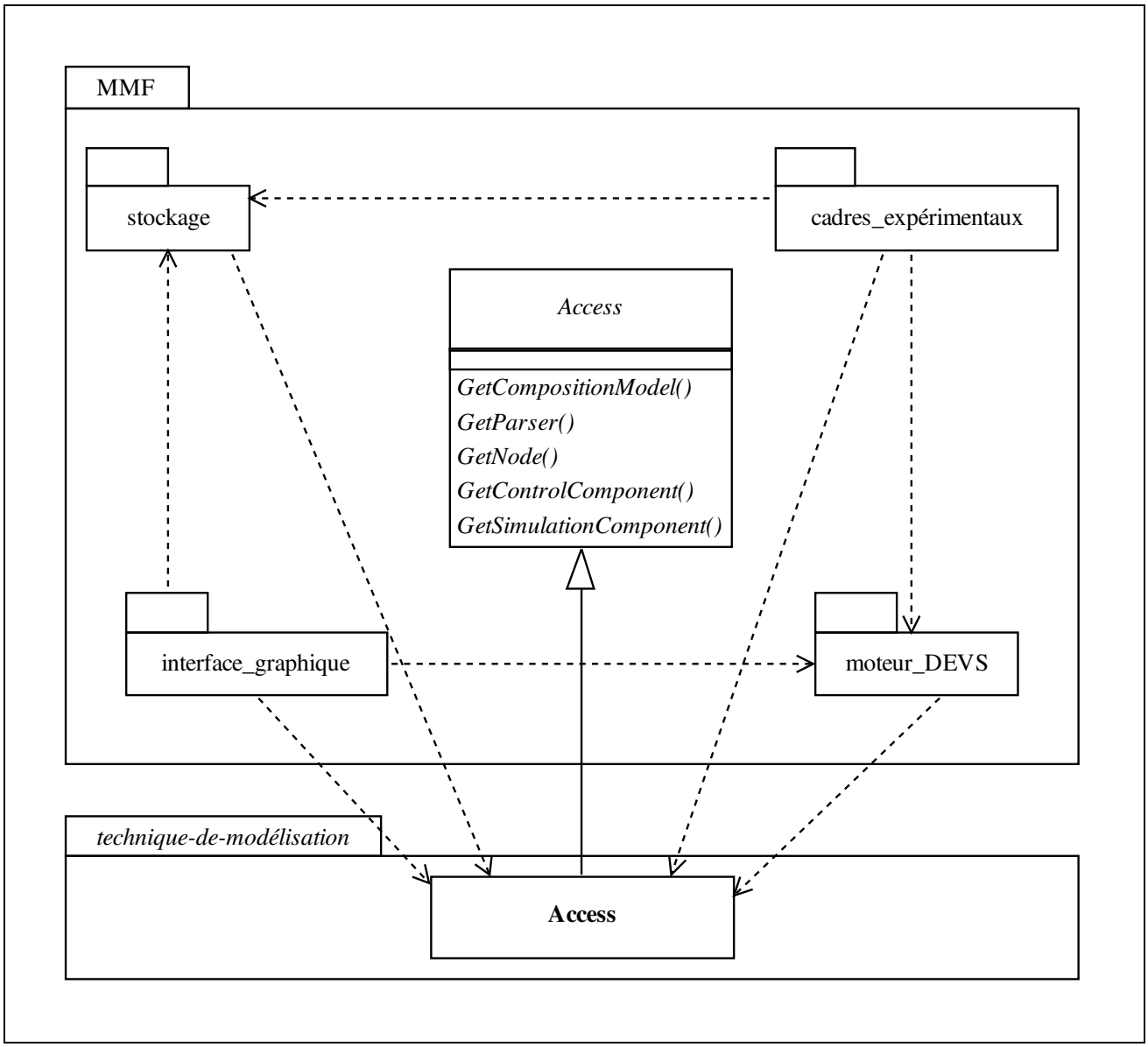

FIG. 3.2*: Vue d'ensemble des packages du cadriciel*

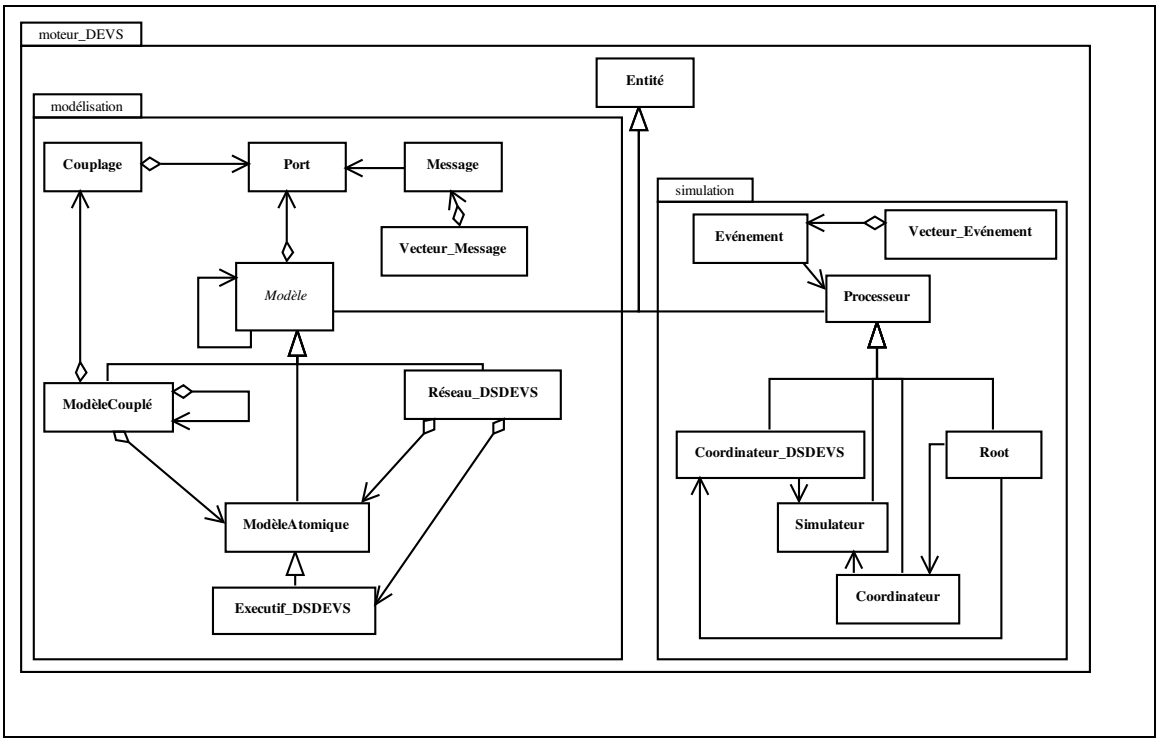

FIG. 3.3*: Le package moteur-DEVS*

ciel puis les classes devant être redéfinies afin d'assurer l'intégration de nouvelles techniques dans l'environnement.

3.2.1 Le package moteur-DEVS

Le package *moteur-DEVS*, figure 3.3, contient les classes nécessaires à la définition de modèles et à leur simulation. Ce package a été séparé des autres pour simplifier l'intégration éventuelle d'autres bibliothèques DEVS telles que ADevs [Nutaro, 2003] ou devsjava [Sarjoughian et Zeigler, 1998].

L'architecture du package suit l'architecture hiérarchique définie dans le formalisme DEVS [Zeigler, 2000] et permet également la définition de modèles à structure dynamique DSDEVS par l'utilisation d'un composant appelé exécutif qui gère les connexions interconnexions entre les modèles. Nous renverrons le lecteurs à [Barros, 1997] pour une explication en détail de cette technique. Un formalisme autorisant la définition de modèles à structure dynamiques est nécessaire pour l'intégration de techniques spécifiques comme nous le verront en chapitre 4.

Le package *moteur-DEVS* est composé de deux sous-packages :

– le sous-package *modélisation* : il contient les classes permettant la modélisation de modèles DEVS et DSDEVS. Nous pouvons noter que dans ce package, la classe **Modèle** peut comprendre des objets de la classe **Port** ; cette agrégation est essentielle pour assurer l'interconnexion de modèles utilisant des techniques de modélisation différentes. En effet tous les couplages de modèles s'effectuent par des liens entre des ports. Chaque objet de la classe **Modèle** communique avec les autres modèles par la médiation d'objets de la classe **Message** : ces objets **Message** étant identiques quelle que soit la technique utilisée.

– Le sous-package *simulation* : il contient les classes implémentant les simulateurs abstraits des modèles DEVS et DSDEVS. Chaque objet de la classe **Modèle** du sous-package *modélisation* est associé à un objet de la classe **Processeur** associée. Le passage d'informations se fait grâce aux objets de la classe **Evènement** contenus dans des listes de la classe **Vecteur-Evènement**.

Tous les modèles créés à l'aide de l'environnement sont identifiés en spécialisant les classes d'un des packages de techniques de modélisation. Les modèles ainsi spécifiés héritent de la classe **ModèleAtomique**, **ModèleCouplé** ou **Réseau-DSDEVS**. Grâce à ces composants de plus haut niveau de spécification et aux interfaces de modèles basées sur les ports, il est possible d'utiliser une interface graphique générique pour aider à la composition de multi-modèles. Le package *interface-Graphique* présenté dans la section suivante est composé des éléments de base permettant la multi-modélisation.

3.2.2 Le package interface-graphique

Le package *interface-graphique* contient les composants graphiques permettant la composition visuelle et interactive de modèles. La figure 3.4 présente les classes composant ce package. A chaque objet de la classe **Composant-Modèle** correspond un objet de la classe **Panel** permettant de modifier ses propriétés (états courants, nom d'instance). Les objets de la classe **Composant-Modèle** sont contenus et affichés par la classe **Espace-de-Travail**, qui est le conteneur graphique de plus haut niveau ; il contient notamment toutes les méthodes d'interaction utili-

sateur (ajout, sélection, déplacement et couplage de modèles). Deux classes spécialisent la classe **Composant-Modèle** : La classe **Composant-Modèle-Composition** et la classe **Composant-Modèle-Atomique**. Les **Composant-Modèle-Composition** sont des conteneurs graphiques pour tous les sous-modèles du modèle associé au composant, ils affichent récursivement tous les sous-modèles (atomiques et/ou couplés) qu'ils contiennent. Les **Composant-Modèle-Atomique** ont eux un comportement spécifique qui ne peut être défini qu'à l'aide d'un éditeur spécialisé. Le composant **Bibliothèque** est le composant graphique permettant l'ajout de composants déjà défini et le stockage des modèles créés. La classe **Application** définit l'interface utilisateur telle qu'elle sera utilisée par le modélisateur et contient un objet de la classe **Espace-de-Travail** lorsqu'un un modèle est chargé en mémoire.

Tous les composants définis dans les packages spécifiques aux techniques doivent hériter des composants du package *Interface-Graphique*. Ils possèdent donc tous une représentation graphique unifiée à haut niveau de description. Cette représentation permet la multi-modélisation par le couplage de modèles de techniques hétérogènes en définissant des liens entre les ports de ces modèles visibles à ce haut niveau de description.

Les objets de la classe **Application** vont rechercher à l'initialisation les packages de techniques disponibles (méthode *GetAvailableTechniques()*). Une liste contenant un objet de la classe **Access** par package spécifique aux techniques est ensuite chargés en mémoire pour acceder à ces sous-systèmes. Pour instancier ces objets, il est fait appel à un objet **AccessLoader** du package *stockage* qui définit les classes permettant le stockage et la récupération de modèles.

3.2.3 Le package stockage

Le package *stockage*, figure 3.5, contient les classes permettant de formater et de lire les modèles. Dans un cadriciel qui se veut générique, il nous fallait choisir le format de stockage le plus ouvert possible. Nous avons donc choisi le langage de balise XML (eXtended Markup Language) comme format de stockage. L'utilisation du langage XML, en plus d'être devenu un standard de stockage *de-facto* dans le monde du génie logiciel, facilite le formatage hiérarchique des modèles.

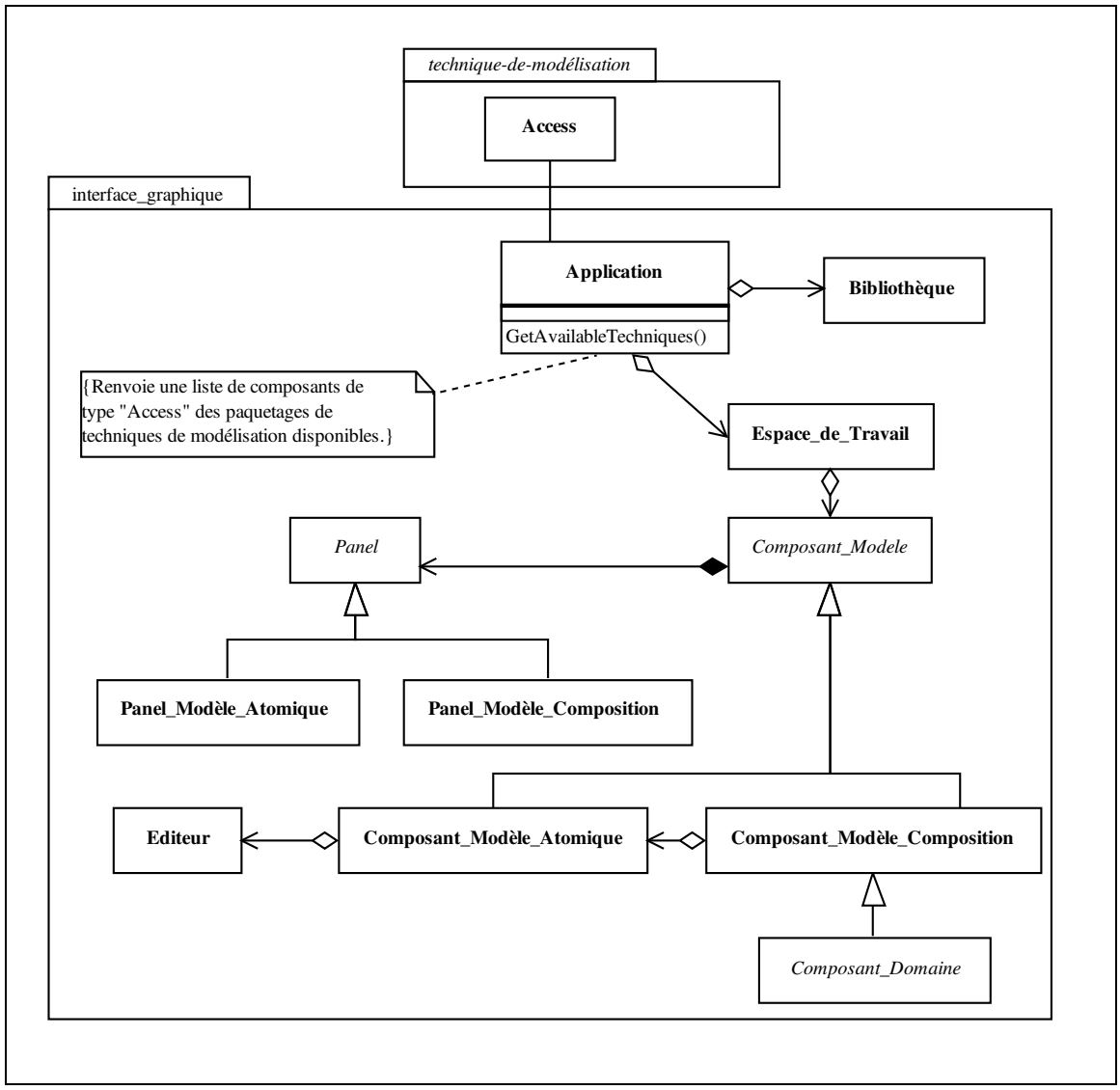

FIG. 3.4*: Le package interface-Graphique*

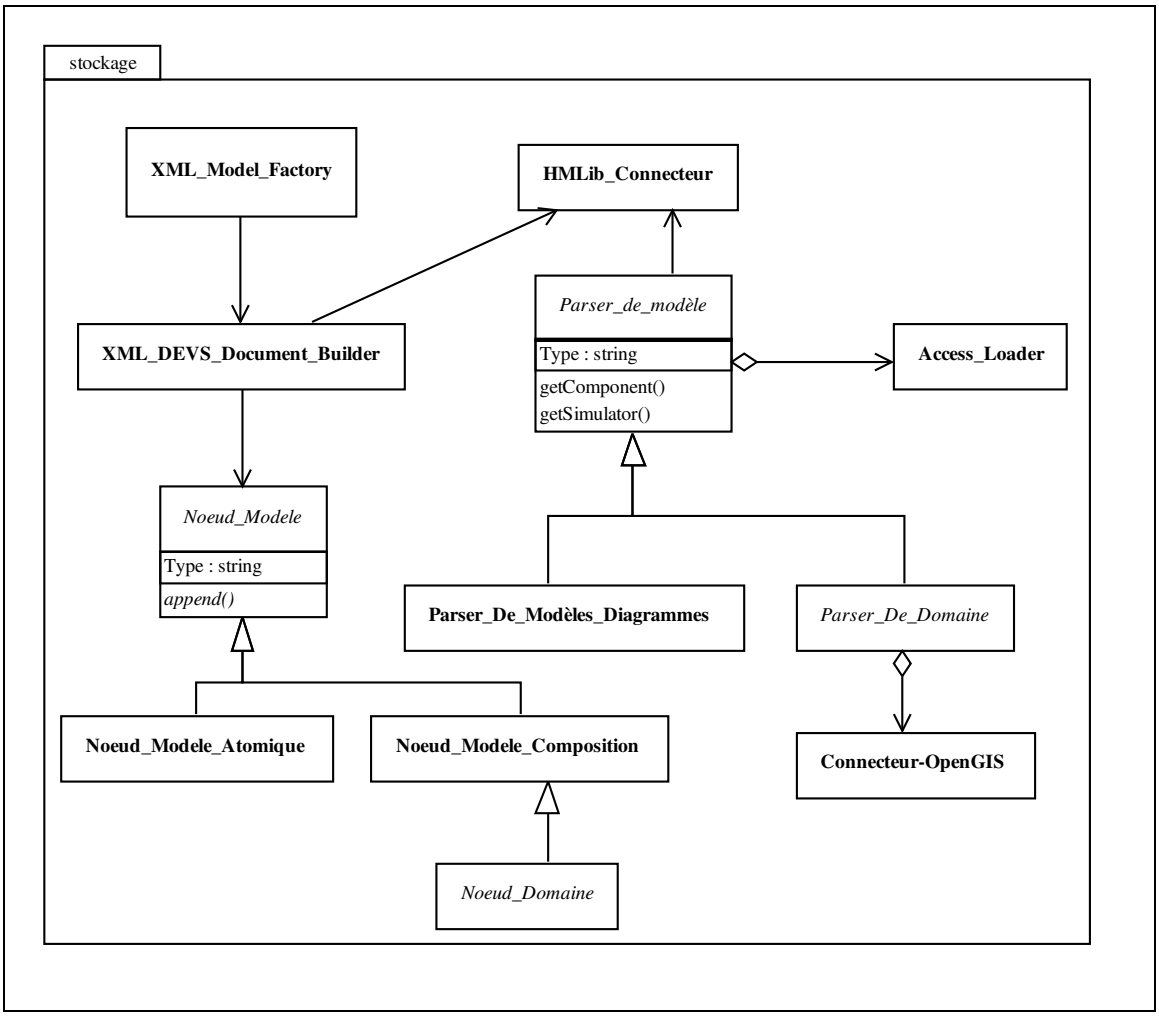

FIG. 3.5*: Le package stockage*

De plus, il nous permet d'assurer sa compatibilité avec le système de gestion de bibliothèque de modèles ***HmLib*** développé au sein de notre laboratoire [Bernardi et al., 2001].

XML est un langage basé sur les balises ou la structure du document (balises, cardinalités et relations) est définie strictement, soit à l'aide d'un schéma XML (XML Schema), soit à l'aide d'un DTD (Document Type Définition ou définition du type de document). Le lecteur intéressé trouvera dans [Means et Harold, 2001] une explication détaillée du format de stockage XML. La structure de fichier devant définir les modèles DEVS n'est toutefois pas encore standardisé à l'heure de la rédaction de ce document. En attendant les propositions de formats étudiés par le DEVS standardization Group [DEVS-Group, 2003] pour le SISO (Society International Standards Organization) [SISO, 2003], nous avons développé notre propre DTD (figure 3.6) pour permettre le stockage de

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE MODEL [
      <!ELEMENT MODEL (TYPE, NAME, CLASS?, BOUNDS?, INPUT*,
           OUTPUT*, CHILD*, EIC?, EOC?, IC?, EXECUTIVE?)>
           <!ELEMENT NAME (#PCDATA)>
           <!ELEMENT TYPE (#PCDATA)>
           <!ELEMENT CLASS (URI)>
           <!ELEMENT BOUNDS (LOCX, LOCY, WIDTH, HEIGHT)>
           <!ELEMENT LOCX(#PCDATA)>
           <!ELEMENT LOCY (#PCDATA)>
           <!ELEMENT WIDTH (#PCDATA)>
           <!ELEMENT HEIGHT (#PCDATA)>
           <!ELEMENT PORT (URI?, #PCDATA)>
           <!ELEMENT URI (#PCDATA)>
           <!ELEMENT CHILD (MODEL* | URI*)>
           <!ELEMENT INPUT (PORT*)>
           <!ELEMENT OUTPUT (PORT*)>
           <!ELEMENT LINK (PORT, PORT)>
           <!ELEMENT EIC (LINK*)>
           <!ELEMENT EOC (LINK*)>
           <!ELEMENT IC (LINK*)>
           <!ELEMENT EXECUTIVE (MODEL* | URI*)> ]
>
```

FIG. 3.6*: Document de définition de format XML des modèles couplés*

ces modèles. Cette DTD ne contient pas de balise de type `ATTLIST`, devant contenir les propriétés des éléments XML ; nous avons volontairement omis de séparer ici, propriétés et données en attendant le travail de standardisation du format DEVS en XML.

Dans cette DTD XML la balise NAME est le nom du modèle, la balise TYPE contient le type de modèle qui est définit par le fichier (Atomique, couplé, DSDEVS network, et tout autres types de modèles résultant de l'ajout de techniques). Dans le cas d'un modèle atomique, CLASS est le lien URI (Uniform Ressource Identifier) vers la ressource contenant, la balise BOUNDS est utilisée uniquement par l'interface graphique pour dessiner le modèle dans un canevas, INPUT est l'ensemble des ports d'entrée, OUTPUT l'ensemble des ports de sortie. La balise CHILD est l'index des composants du modèle couplé (dans l'ordre de priorité) ces composants sont, soit décrits récursivement dans des balises de type MODEL, soit par un lien (balise URI) vers le fichier XML correspondant. Les couplages sont définis par des balises LINK prenant un couple de PORT précédés par le lien URI du modèle auxquels ces ports appartiennent si il ne s'agit pas de ports concernant le modèle défini par le fichier. EIC est l'ensemble des couplages d'entrée, IC est l'ensemble des couplages internes et EOC est l'ensemble des couplages de sortie. Dans le cas d'un modèle DSDEVS, la balise EXECUTIVE contient le modèles de l'exécutif de réseau DSDEVS qui est décrit soit directement dans une balise de type MODEL, soit par un lien URI.

Cette définition correspond au format de fichier utilisé par le cadriciel. Pour répondre à des besoins de stockage spécifiques cette définition peut être étendue en ajoutant des balises mais jamais en n'en enlevant ; ceci permet d'assurer une compatibilité ascendante du format de fichier. L'accès et la création de ces fichiers requièrent les classes présentées en figure 3.5. Pour permettre le stockage et la récupération de modèles, deux architectures distinctes sont nécessaires :

– Pour permettre le stockage, nous avons utilisé le patron de conception ***Builder***, qui offre la possiblité de séparer la construction d'un objet complexe de sa représentation, afin que le même processus de construction puisse créer des représentations différentes. La classe **XML-DEVS-Document-Builder** étant ici le pendant du *Director* du patron de conception, il est créé une instance de cette classe pour chaque modèle à stocker. A chaque sous-modèle est agrégé un objet **Noeud-Modèle** (équivalent du *builder* du patron de conception), cet

objet contient la méthode *append()* permettant de générer le code XML contenant toutes ses propriétés. Un champ *Type* renseigne alors sur la technique de modélisation utilisée. Dans le cas d'un modèle atomique, c'est un lien *URI* contenant l'adresse du fichier de code qui est agrégé dans la balise CLASS. Dans le cas d'un modèle de composition la méthode *append()* est appelée récursivement sur tous les sous-modèles. Cette méthode doit être surchargée si la technique de modélisation nécessite le stockage d'informations spécifiques ; par exemple, les modèles devant stocker des structures spatiales devront contenir un objet de type Noeud héritant de **Noeud-Domaine**. Les objets de la classe **XML-DEVS-Document-Builder** sont les conteneurs de la structure et des données générées par les objets de type Noeuds : ils sont stockés dans une bibliothèque de modèles à l'aide d'un objet de la classe **HMLib-connecteur**.

– La récupération de modèles se fait à l'aide d'objets de la classe **Parser-de-modèles**, connectés à une bibliothèque HMLib à l'aide de la classe **HMLib-Connecteur**. Chaque technique de modélisation doit définir un parser spécifique, ce qui permet d'instancier automatiquement les parsers appropriés dans le cadre d'un multi-modèle en analysant le champ *type* de chaque Noeud MODEL du fichier XML. Le champ TYPE correspond au package à utiliser pour instancier le parser approprié, ces instanciations se font grâce à la classe **Access-Loader** qui instancie les objets **Access** pour chaque package de technique de modélisation. Ces objets du type **Access** renvoient ensuite un lien vers la classe **Parser** appropriée : soit un parser générique comme celui de la classe **Parser-De-Modèles-Diagrammes**, soit un parser spécifique défini dans le package de la technique. Les parsers déclenchent des actions spécifiques si leur utilisation est sollicitée dans le cadre d'une expérimentation (génération des simulateurs en invoquant la méthode *getSimulator()*), ou dans le cadre de la modélisation (génération des composants graphiques dans l'interface en invoquant la méthode *getComponent()*).

Une fois les modèles stockés, ils peuvent être simulés dans un cadre expérimental. Chaque technique de modélisation requiert néanmoins le développement d'un cadre expérimental spécifique. Il est donc nécessaire de définir une architecture générale à laquelle devront être soumis ces

cadres spécifiques pour assurer leur interopérabilité. C'est le but du package *cadres-expérimentaux* décrit dans la section suivante.

3.2.4 Le package cadres-expérimentaux

Le package *cadres-expérimentaux*, figure 3.7, contient les classes de bases pour permettre la simulation de multi-modèles à l'intérieur d'une interface homogène.

Il contient les classes nécessaires à la visualisation et au contrôle de la simulation. Comme pour les **Composants-Modèle** du package **interface-graphique**, chaque **Processeur** du package *moteur-DEVS* est associé à un composant graphique **Composant-Processeur**, capable d'afficher les propriétés du modèle associé au processeur en cours de simulation. Ainsi, pendant la simulation, l'état des modèles atomiques est affiché par un **Composant-Simulateur** et celui des modèles de composition par un **Composant-Coordinateur**.

Le cas particulier des modèles spatialisés nécessite la création d'une classe spécifique **Composant-Domaine** capable d'afficher les propriétés spatiales des modèles. La gestion des propriétés spatiales, comme leur représentation, sont dépendantes de la technique de modélisation utilisée. Il est donc impossible de définir des méthodes standards de haut niveau pour cette classe qui reste abstraite dans le cadriciel. Les propriétés spatiales de tous les **Composant-Domaine** en cours de simulation ont un **Gestionnaire-Spatial** agrégé qui centralise l'accès aux données géographiques pour faciliter leur traitement.

Les objets de la classe **Espace-Simulation** sont les conteneurs pour les objets **Composant-Processeur**. La classe **Espace-Simulation** est connectée au **Composant-Processeur** attaché au processeur de plus haut niveau de l'arbre de simulation (Classe **Root** du package *moteur-DEVS*). Chaque objet du type **Espace-Simulation** possède un **Controlleur-de-Simulation** qui est attaché au processeur de plus haut niveau (Root) pour lui envoyer tous les événements d'entrée et récupérer tous les événements d'entrée/sortie générale du modèle dans un objet de type **Logger**.

La classe **Espace-Simulation** est chargée des interactions utilisateur et récupère les événements d'entrée (clavier et souris). Ces événements sont, traités par le **Composant-Controle**, asso-

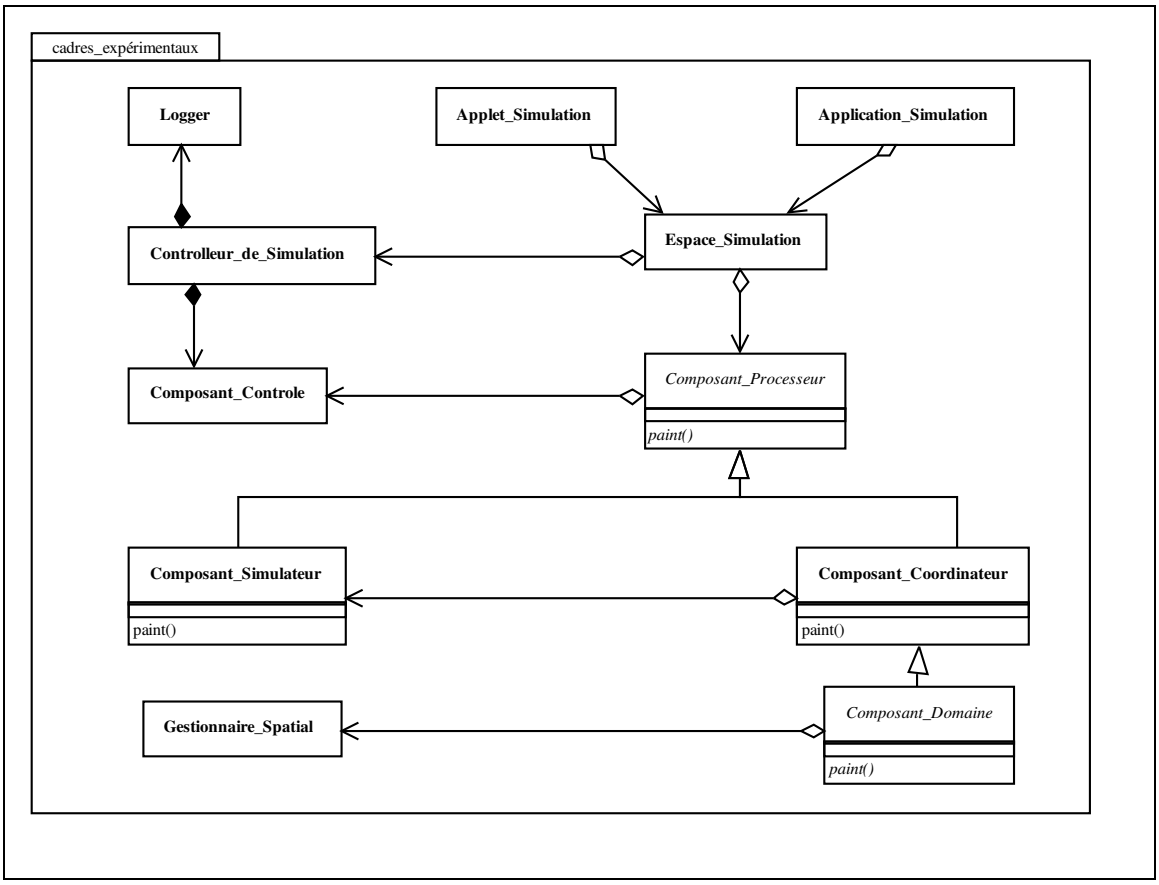

FIG. 3.7*: Le package cadres-Expérimentaux*

ciés à chaque **Composant-Processeur** et dépendants de la technique utilisée qui détermine comment transformer ces interactions en événements d'entrée du modèle.

Dans le cadre d'une application scientifique de la simulation, les objets **Espace-Simulation** peuvent être agrégés dans des objets du type **Application-Simulation** qui offrent la possibilité de stocker les fichiers de sortie pour un post-traitement éventuel. Dans le cadre d'une application en vue de la prise de décision, les **Espace-Simulation** peuvent être agrégés à des **Applets-Simulation** et rendus disponibles sur le web pour être utilisés plus facilement par les personnes en charge de ces décisions.

La visualisation de modèles, utilisant en cours d'exécution, différentes techniques en parallèle, est facilitée par la classe **Composant-Processeur** qui offrent un canevas unique pour l'affichage. La méthode d'affichage (*paint()*) peut ainsi être appelée récursivement sur tous les sous-processeurs. Toutefois, avant d'être affichée une technique doit être définie dans un package. La section suivante montre comment intégrer une technique en prenant l'exemple de la technique la plus simple à intégrer dans le cadriciel : classic-DEVS.

3.2.5 Le package classic-DEVS

La figure 3.8 présente le package classic-DEVS tel qu'il est défini pour permettre l'intégration de la technique de modélisation DEVS et de la simulation visuelle (où l'état des est représenté graphiquement en cours de simulation) et interactive des modèles ainsi créés.

Tous les composants de base permettant la modélisation de modèle DEVS sont déjà présents dans le cadriciel. Intégrer classic-DEVS dans le cadriciel revient donc simplement à créer une classe **classic-DEVS.Access** dérivant de la classe abstraite **Access** du package général, puis, à implémenter les méthodes pour renvoyer les liens vers les composants adéquats (figure 3.8). Dans le cas de techniques de modélisation plus complexes, comme nous l'envisagerons dans le chapitre suivant, de nombreuses classes devront encore être créées : les liens des objets de la classe **Access** associés à la technique de modélisation ne seront donc plus établis vers des classes du cadriciel mais vers de nouvelles classes internes aux packages de techniques de modélisation.

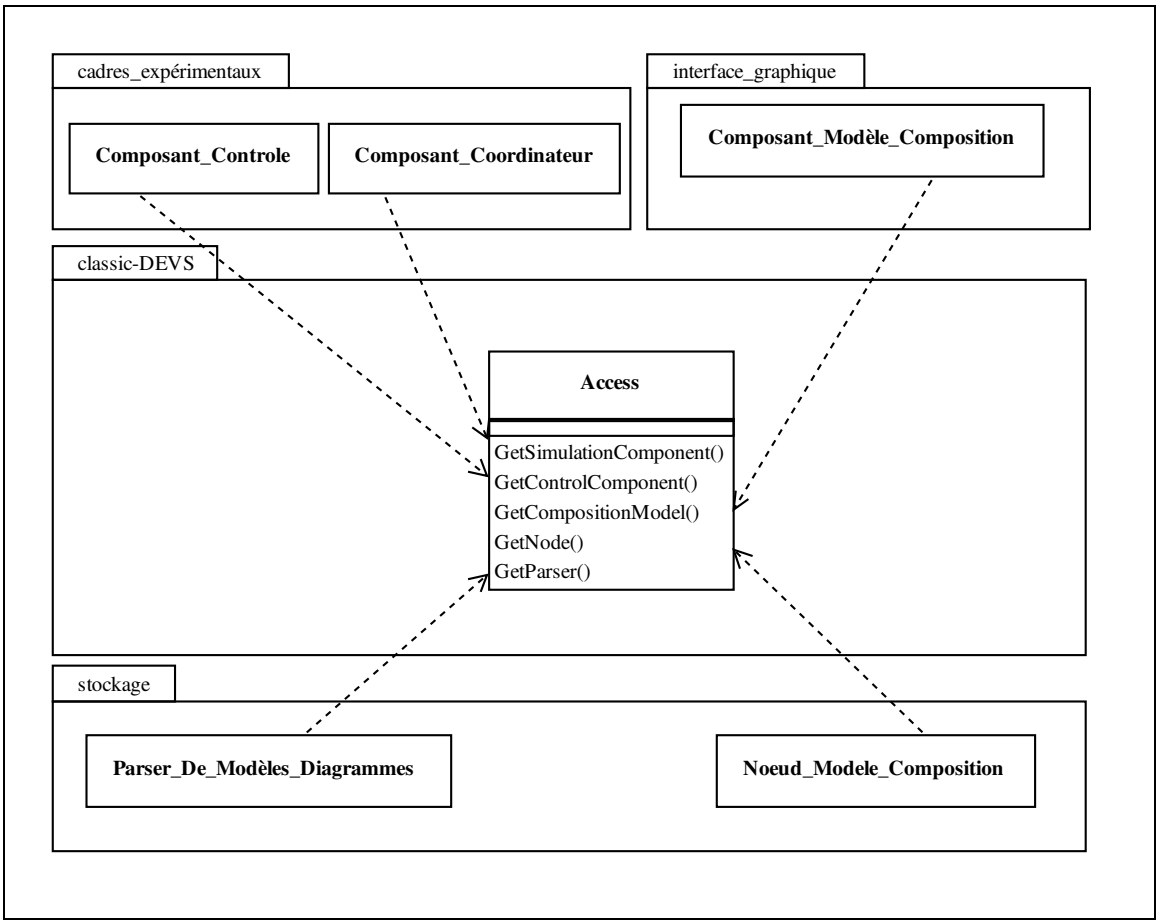

FIG. 3.8*: Le package classic-DEVS*

3.3 Conclusion

A l'issue de ce chapitre nous pouvons faire un point sur la conduite de ce travail. Nous avons présenté un cadriciel de multi-modélisation destiné à l'étude de systèmes complexes par la simulation. Ce cadriciel possède une architecture modulaire et est basé sur l'utilisation de packages susceptibles d'évoluer sans compromettre l'intégrité de l'ensemble.

Le principal intérêt du cadriciel est d'être suffisamment ouvert pour intégrer de nouvelles techniques de modélisation tout en gardant les classes constitutives du package de base, intactes.

De plus, de part son découpage en quatre partie distinctes, il est possible de répartir le travail de conception et d'amélioration de chaque package indépendamment.

La quantité des techniques disponibles dans le cadriciel lui conférera son universalité. Ceci nous conduit à exposer dans le chapitre suivant comment nous avons cherché à intégrer différentes techniques de modélisation à l'environnement logiciel, tant au niveau formel qu'à celui de la conception orientée objet.

CHAPITRE 4

Intégration de techniques dans le cadriciel

Le chapitre précédent a présenté l'architecture générale du cadriciel et les choix de conception. L'intérêt du cadriciel est d'être ouvert et de fournir une base théorique forte pour permettre la réutilisation, le stockage et le couplage simplifié de modèles hétérogènes. Toutefois, sans extensions, ce cadriciel ne permet d'étudier qu'un nombre limité de systèmes car il n'est pas possible de conceptualiser directement des modèles grâce à des paradigmes de modélisation spécifiques. Ce cadriciel ne constitue que la base pour l'ajout de techniques d'études de systèmes dans un cadre de multi-modélisation. Ce chapitre présente l'ajout de techniques appliqués au cadriciel pour permettre à un modélisateur de conceptualiser les modèles selon différents paradigmes. La liste de ces techniques n'est pas exhaustive mais le reflet des problèmes induits par la modélisation informatique dans notre laboratoire *systèmes physiques pour l'environnement*. Après une brève introduction sur les paradigmes de simulation, ce chapitre présente ces techniques et détaille particulièrement les techniques originales développées spécifiquement dans le cadre de cette étude.

4.1 Les paradigmes de modélisation

Les paradigmes de modélisation sont des métaphores informatiques que le modélisateur emploie pour conceptualiser le monde. Ils proposent un cadre logique pour la description de problèmes et permettent ensuite l'analyse des solutions par la simulation. Le grand nombre de paradigmes utilisés aujourd'hui dans le domaine de la modélisation de systèmes physiques complexes pose toutefois le problème de la cohérence au niveau syntaxique et conceptuel d'un multi-modèles utilisant un sous-ensemble de ces méthodes. Ainsi on décrit les paradigmes comme étant : basés sur des processus ou basés sur des agents, spatialement explicites ou non, variants ou invariants en structure, déterministes, stochastiques ou multi-échelles ou encore utilisant une échelle unique en espace et en temps. Nous nous limitons dans le cadre de cette étude aux paradigmes permettant de spécifier des modèles causaux (dont les sorties sont la conséquence d'une entrée) et déterministes (à une entrée donnée ne peux correspondre qu'une seule sortie). A chaque paradigme correspond une ou plusieurs techniques de modélisation et de simulation, permettant de spécifier les modèles qui utilisent la métaphore proposée par le paradigme. Dans le cadre de notre étude, deux grandes familles de techniques de modélisation se détachent : celle spécifique aux modèles spatialisés (dont les propriétés évoluent dans l'espace à travers le temps) et celle propre à ceux qui ne le sont pas. Nous distinguons ces deux types de modèles parce qu'ils nécessitent des moyens de traitement de résultats et d'observation complètement différents.

La grande majorité des systèmes étudiés au sein de notre laboratoire sont des systèmes de dynamique de l'environnement et en conséquence, de dynamique spatiale. Toutefois, avant de réfléchir aux modes de représentation de dynamique spatiale, il est important de définir les différents formats de stockage de l'information spatiale. Comme celles relatives au temps, les propriétés d'un système à travers l'espace peuvent être représentées de manière continue ou discrète. Les représentations continues utilisent des méthodes analytiques basées sur la physique, comme par exemple le modèle de la vitesse de chute des corps. La prise en compte discrète de l'espace se fait généralement par l'utilisation de rasters, mais l'étude de certains systèmes peut nécessiter l'utilisation

de cartes vectorielles ou de modèles numériques de terrain en trois dimensions. Pour les modèles non-analytiques, plusieurs techniques de modélisation ont été développées :

- Les modèles cellulaires [Wainer et Giambiasi, 2001] où l'espace est représenté sous la forme d'une grille, chaque maille figurant un modèle de l'espace qu'elle occupe ; ces modèles peuvent être adaptés pour la résolution d'équations différentielles ordinaires.
- Les modèles multi-agents, [Wooldridge, 2002], où les modèles correspondent à des entités pouvant se déplacer, entrer en relation et communiquer dans un espace (carte raster ou vectoriel).
- Les modèles à propagation d'interface [Dafermos, 1972], [Risebro et Tveito, 1992], où l'on s'intéresse à l'évolution de l'enveloppe d'un phénomène dans un espace (carte raster ou vectoriel).

Dans le cadre de l'étude de systèmes ne présentant pas de dynamique spatiale, la technique la plus couramment utilisée est celle des "équations différentielles ordinaires". Mais l'utilisation d'équations différentielles requiert une parfaite connaissance du système physique à étudier. Toutefois, des techniques empiriques de modélisation (régression, réseaux de neurones, ...) peuvent permettre de passer outre cette limitation si des données d'observation sont disponibles en quantité suffisante. De tels modèles réalisent l'apprentissage de la dynamique du système à modéliser et sont très utilisés dans le domaine de la modélisation environnementale. Pour pouvoir assurer l'interoperabilté et l'intégration de modèles utilisant une telle diversité de techniques, nous devons nous assurer qu'ils partagent le même formalisme de base, DEVS.

Le formalisme DEVS classique ne permet que la spécification de systèmes invariants en structure, déterministes et n'ayant pas de propriétés spatiales. Toutefois, si le formalisme est suffisamment complet pour permettre la définition de tous ces types de systèmes, l'ajout de différentes techniques de modélisation lui est nécessaire avant de permettre leur spécification. Adapter une technique consiste à décrire l'ensemble de ses composants et leurs dynamiques, à l'aide des règles du formalisme DEVS. Un grand nombre de travaux portent sur ces adaptations comme ceux sur les automates cellulaires [Wainer et Giambiasi, 2001], la logique floue [Kwon et al., 1996] ou les multi-agents [Schattenberg et Uhrmacher, 2001]. Toutefois la base DEVS classique est trop res-

trictive pour permettre l'adaptation directe de paradigmes plus spécifiques, il est alors nécessaire de généraliser DEVS. Il existe plusieurs travaux de généralisation de DEVS, notamment le formalisme GDEVS [Giambiasi et al., 2002] qui autorise la modélisation hybride continue/discrète de systèmes ou encore le formalisme DSDEVS [Barros, 1997] permettant la spécification de modèles à structure dynamique. Ces généralisations garantissent néanmoins une compatibilité ascendante avec le formalisme DEVS car elles ont la propriété de *fermeture sous couplage* ainsi que des interfaces basées sur la notion de ports. Les sections suivantes présentent en détail les techniques intégrées à l'environnement, les cas d'applications de ces techniques, leurs spécifications DEVS et les packages permettant de les ajouter au cadriciel.

4.2 Modèles avec feedback

Une des limitations de l'utilisation de DEVS est que le modélisateur doit avoir une bonne connaissance du comportement physique du système à étudier pour pouvoir obtenir des résultats tangibles. Dans le cas contraire, des modèles empiriques dits "auto-apprenants" sont fréquemment utilisés pour l'étude de systèmes physiques. Des méthodes adaptatives sont mises en place dans de tels modèles pour capturer la dynamique du système à partir d'un ensemble de données empiriques représentatives. De tels modèles utilisent un retour d'information ou *feedback*, pour adapter leurs comportements à de nouvelles situations. Un modèle adaptatif peut ainsi *apprendre* le comportement d'un système simple en adoptant d'abord une dynamique grossièrement semblable à elle de ce système, puis en affinant son comportement en réaction à des retours d'information sur la différence existant entre le modèle et le système réel. Les "réseaux de neurones artificiels" sont les plus connus des algorithmes adaptatifs et sont utilisés avec succès pour la simulation de nombreux systèmes complexes.

4.2.1 Cas d'utilisation de Feedback-DEVS

Les modèles empiriques sont souvent préférés aux modèles théoriques car ils sont plus simples à mettre en oeuvre et souvent plus performants lorsque les données disponibles sont de bonne qua-

lité. Néanmoins ils ne présentent pas l'avantage de pouvoir être prouvés théoriquement. Les modèles empiriques sont des boîtes noires dont toutes les réactions aux stimuli externes ne peuvent être prévues et peuvent donc présenter des aberrations. Un modèle théorique, lorsqu'il existe, nécessite une quantité importante de paramètres pour fournir un résultat précis dans le cadre d'une simulation numérique. En prenant en compte ces limitations, nous avons défini trois cas d'utilisation de modèles adaptatifs.

- La simulation concurrente : elle est utilisée lorsqu'on dispose d'un modèle empirique dont les résultats ont été validés avec plus de succès que le modèle théorique mais que la validité des résultats de simulation sont critiques. Un modèle de validation établi théoriquement est alors utilisé pour éliminer les éventuelles aberrations du modèle empirique [Stimpfl-Abele, 1995]. Le modèle théorique de validation ne peut pas donner de réponses précises, souvent par manque de paramètres, mais peut donner un intervalle de confiance. Ce type de modèle impose que l'apprentissage du modèle adaptatif ait été réalisé avant la phase de simulation. La Figure 4.1 montre un modèle adaptatif empirique dans le cadre d'une utilisation en simulation concurrente. Le modèle distributeur copie les événements d'entrée aux deux modèles (adaptatif et de validation), puis les sorties sont comparées par un "Comparateur". Le "Comparateur" vérifie que les valeurs de sortie du modèle empirique se situent bien dans un intervalle de confiance fourni par le modèle de validation avant de générer une sortie, dans le cas contraire la sortie peut être un message d'erreur ou une valeur approximative calculée à partir de l'intervalle du modèle de validation.
- Les modèles adaptatifs comme sous-modèles : ils sont utilisés dans le cas de l'étude d'un système complexe, une partie du modèle ne peut être modélisée de manière classique. Ce cas d'utilisation nécessite l'apprentissage préalable du modèle empirique. La figure 4.2 montre l'utilisation d'un modèle adaptatif dans le cadre de son utilisation en sous-modèle. Ici, le modèle adaptatif est couplé en entrée à un sous-modèle A et en sortie à un sous-modèle B.
- Les modèles adaptatifs pour prendre en compte des changements en cours de simulation : ils sont utilisés dans le cas d'une simulation où le comportement du modèle doit s'adapter en temps réel à un changement du système qu'il représente. La figure 4.3 montre l'utilisation

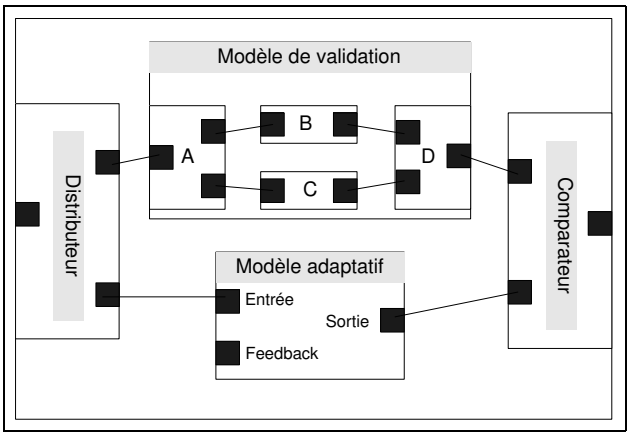

FIG. 4.1*: Utilisation de modèles adaptatifs dans le cadre de simulation concurrente*

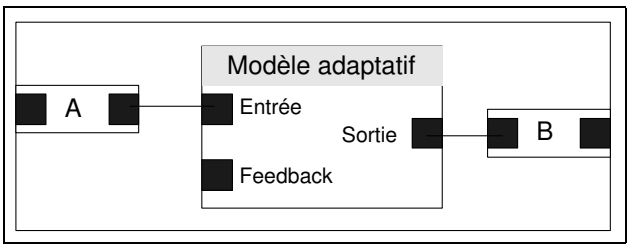

FIG. 4.2*: Utilisation de modèles adaptatifs comme sous-modèles*

du modèle adaptatif dans le cadre de la prise en compte de changements comportementaux. Le modèle adaptatif présente alors une entrée feedback par laquelle peut transiter le retour d'information sur les différences existant entre les résultats de la simulation et les résultats de mesure d'expérimentation. Un tel modèle peut donc adapter son comportement en cours de simulation et il n'est alors pas nécessaire de réaliser l'apprentissage avant la simulation.

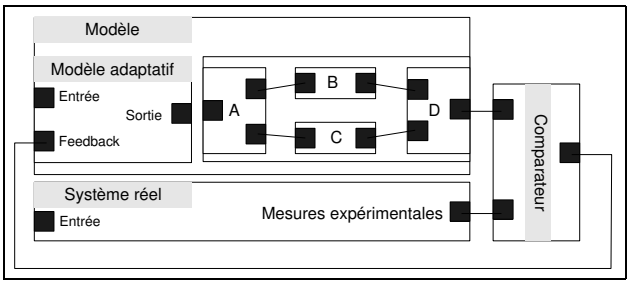

FIG. 4.3*: Utilisation de modèles adaptatifs pour la prise en compte de changements comportementaux en cours de simulation*

4.2.2 Spécification de modèles Feedback-DEVS

Feedback-DEVS s'inspire du formalisme F-DEVS [Kofman et al., 2000] pour permettre la spécification de modèles adaptatifs. F-DEVS a été développé pour la modélisation de fautes, cependant F-DEVS intègre des mécanismes de modification de structure du modèle qu'il n'est pas nécessaire de prendre en compte dans la résolution des problèmes auxquels Feedback-DEVS est destinés.

Un modèle feedback-DEVS est décrit par l'ensemble :

$$\text{FM} = < Xf, S, Y, \delta_{int}, \delta_{ext}, \delta_{react}, \lambda, t_a > \text{Où}$$

- Xf : $\{(p_i, v_i) \bigcup (p_f, v_f) | (p_i \in$ ports d'entrés standards, $p_f \in$ ports d'entrée de feedback, $v_i \in$ Xpi, $v_f \in$ Xpf$)\}$ ensemble des couples entrées/valeurs pour chaque port d'entrée (standard ou feedback),
- S : $S' \bigcup S_f$ est l'ensemble des états internes avec S' l'ensemble des états correspondant à un comportement normal et S_f l'ensemble des états que le modèle pourrait atteindre en cas de réaction à un feedback.
- $\delta_{react} : S \times Xf \rightarrow S$ est la fonction de réaction,

Dans FM $< Y, \delta_{int}, \delta_{ext}, \lambda >$ sont identiques au modèle DEVS standard ; la différence se situe dans une séparation explicite entre les entrées standard et les entrées de feedback. Cette différence permet un traitement explicitement séparé des messages arrivant sur Xpi et Xpf. Les entrées Xpi déclenchent l'envoi de messages "X" tandis que les entrées Xpf déclenchent l'envoi de messages "F". L'algorithme du simulateur abstrait pour modèles feedback-DEVS ne diffère du simulateurs standard que par le comportement à la reception de message de type F qui déclenche la fonction δ_{react} et remet à jour le temps au prochain changement d'état.

Le formalisme feedback-DEVS présenté ici permet l'intégration de modèles à comportements adaptatifs dans le cadriciel de modélisation générique. La section suivante présente le paquetage **feedback-DEVS** qui permet l'ajout de ce type de modèles dans le cadriciel.

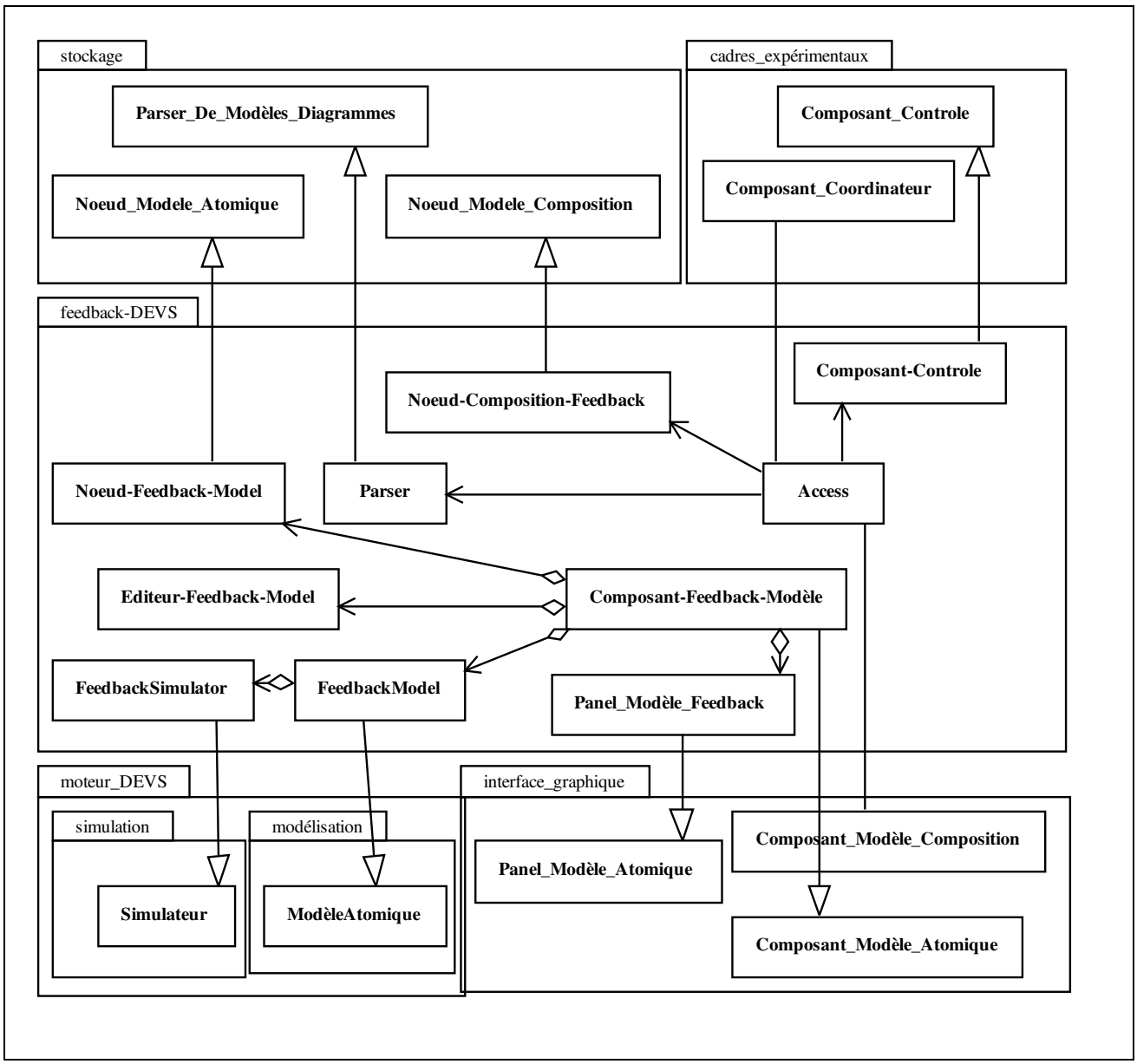

FIG. 4.4*: Le package feedback-DEVS*

4.2.3 Intégration de Feedback-DEVS dans le cadriciel

La figure 4.4 présente le package *feedback-DEVS* contenant les classes spécifiques à cette technique. Comme les liens vers les composants spécifiques sont donnés par une instance de la classe **feedback-DEVS.Access**, il n'est pas nécessaire de modifier les classes des packages du cadriciel. Nous avons vu que les modèles Feedback-DEVS sont couplés à l'intérieur de modèles de composition DEVS classiques, les classes **Composant-Coordinateur** et **Composant-Modèle-Composition** ne sont donc pas redéfinies. Il est cependant nécessaire de développer la classe **Noeud-Composition-Feedback**, différente de la classe **Noeud-Modèle-Composition**, car elle doit permettre le stockage de liens vers des ports d'entrée de type feedback. Le modèle atomique de type feedback est différent du modèle atomique classique, ce qui nécessite l'ajout d'une classe **Feedback-Model** héritant de **AtomicModel** et de la classe simulateur associées **Feedback-Simulator**. L'intégration d'une technique comme feedback-DEVS est facilitée par sa très forte similarité avec la technique classique DEVS directement disponible dans le cadriciel. Dans le cas de l'étude de systèmes distribués spatialialement les données à traiter sont de nature différente. Les sections suivantes présentent les techniques de modélisation pour les problèmes spatialisés, techniques définies pour l'environnement afin de prendre en compte la dimension spatiale des modèles.

4.3 Modélisation par automate cellulaire

Les modélisations les plus courantes de phénomènes spatiaux s'élaborent en affectant aux zones de l'espace dont la surface est identique, un caractère discret : nous les appellerons les cellules. Pour spécifier un modèle cellulaire, il suffit de spécifier le comportement d'une cellule. Chaque cellule doit pouvoir échanger des informations avec les cellules de son voisinage. Le voisinage est constitué habituellement des cellules situées aux points cardinaux mais peut aussi être dynamique et fonction d'autres attributs. Les modèles cellulaires sont les plus courants des modèles spatialement distribués, il était nécessaire de les inclure dans l'environnement. Cependant, la transcription de la technique d'automate cellulaire en DEVS ayant déjà été l'objet de travaux complets, nous n'en présenteront pas en détail les spécifications. Toutefois pour les lecteurs intéressés,

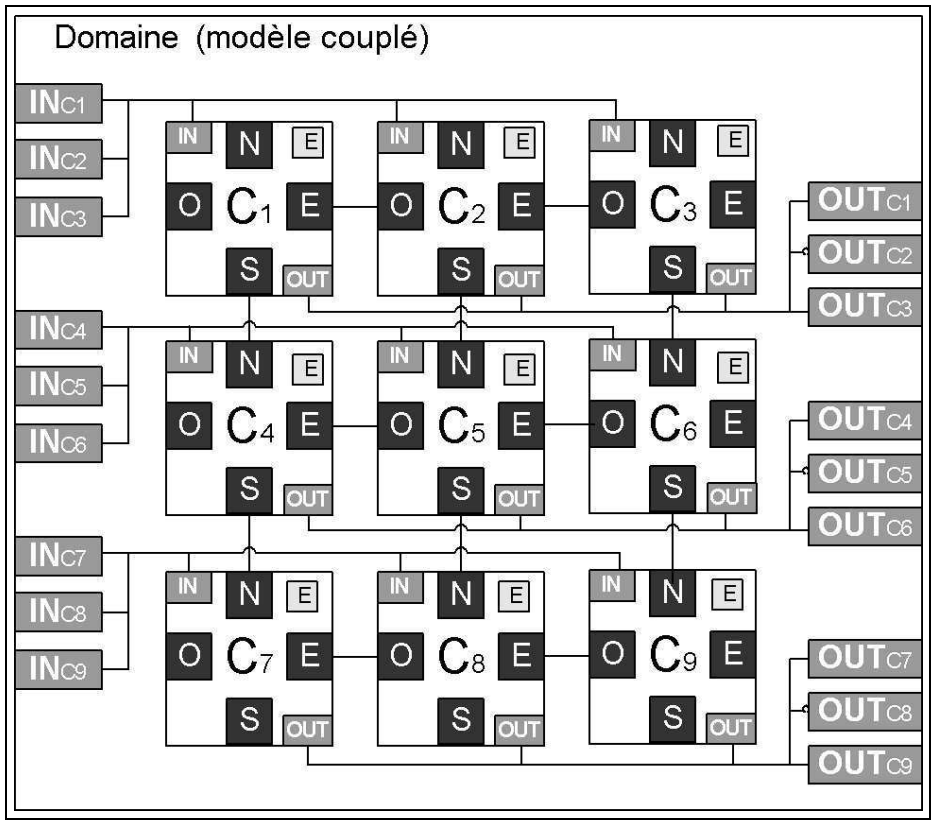

FIG. 4.5*: Modèle couplé représentant un domaine de neuf cellules (C1..9), couplées entre elles par des liaisons de port à port aux points cardinaux (Nord, Sud, Est et Ouest), et possédant un port d'entrée et de sortie chacune (IN et OUT)*

nous conseillons la consultation de [Wainer et Giambiasi, 2001]. Il existe deux formes principales de spécification de modèles cellulaires avec DEVS :

– en considérant chaque cellule comme un modèle atomique. Les couplages internes correspondent alors aux fonctions de voisinage et l'ensemble des cellules corresponds à un domaine. La figure 4.5 présente un modèle couplé comportant neuf cellules. Ce type de spécification est le plus souple car il autorise la manipulation de domaines composés de cellules hétérogènes. Soulignons toutefois que le passage nécessaire par des ports de messages, devienne vite un handicap lors de la simulation.

– En considérant un modèle atomique comme un ensemble de cellules. Chaque cellule est alors un des état de ce modèle et les fonctions de voisinages sont spécifiées par les fonctions de transitions. De tels modèles sont appelés *multicomposants* [Zeigler, 2000]. La figure 4.6 présente un modèle atomique *multicomposants* composé de neuf cellules. Les modèles *mul-*

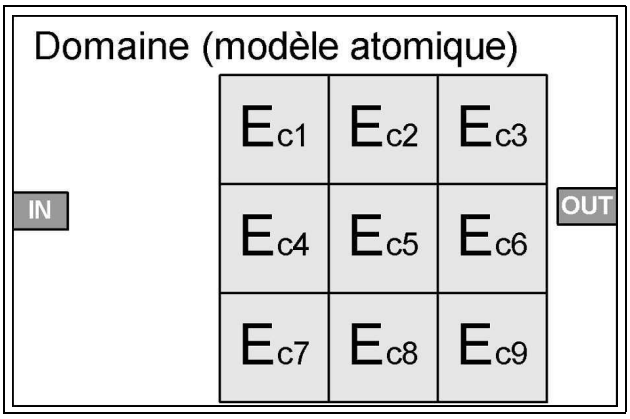

FIG. 4.6*: Modèle atomique* multicomposants *représentant un domaine de neuf cellules (E1..9) possédant un port général d'entrée et de sortie (IN et OUT)*

ticomposants sont limités car constitués d'un ensemble de cellules homogènes, les mêmes règles de transitions s'appliquant pour toutes ces cellules ; toutefois ils ont un avantage en matière de vitesse de simulation car ils peuvent s'affranchir des couplages internes de voisinage et donc des lourds passages de données.

L'intégration de la technique de modélisation cellulaire dans le cadriciel est faite grâce au package *raster-DEVS* présenté en figure 4.7. Ce package contient les classes spécifiques aux techniques de modélisation par automate cellulaire où chaque cellule est un modèle atomique. Dans l'interface graphique de modélisation, chaque domaine que l'on veut représenter par un ensemble de cellules est figuré par un objet **Composant-Modèle-Raster** contenant les différentes catégories de cellules de type **Composant-Cell** possédant un éditeur de code pour l'objet de type **Cell**, héritant de **AtomicModel** et définissant son comportement. Chaque objet **Composant-Modèle-Raster** est également associé à un objet de type **Cell-Domain** héritant de **CoupledModel** ce qui permet de spécifier les couplages internes par structures de maillages plutôt que des interconnexions de ports. Les objets **Cell** et **Cell-Domain** peuvent être simulés par des objets **Simulator** et **Coordinator** standard et ne nécessitent donc pas la création de classes particulières à cet effet. Les modèles cellulaires sont stockés par des objets de type **Noeud-Domaine-Raster** et **Noeud-Cell** et récupérés grâce à un objet **Parser** spécifique héritant de ***Parser-De-Domaine***. Le **Parser** est spécifique car il doit réaliser le chargement d'un **Composant-Domaine-Raster** qui affiche les états des objets **Cell**

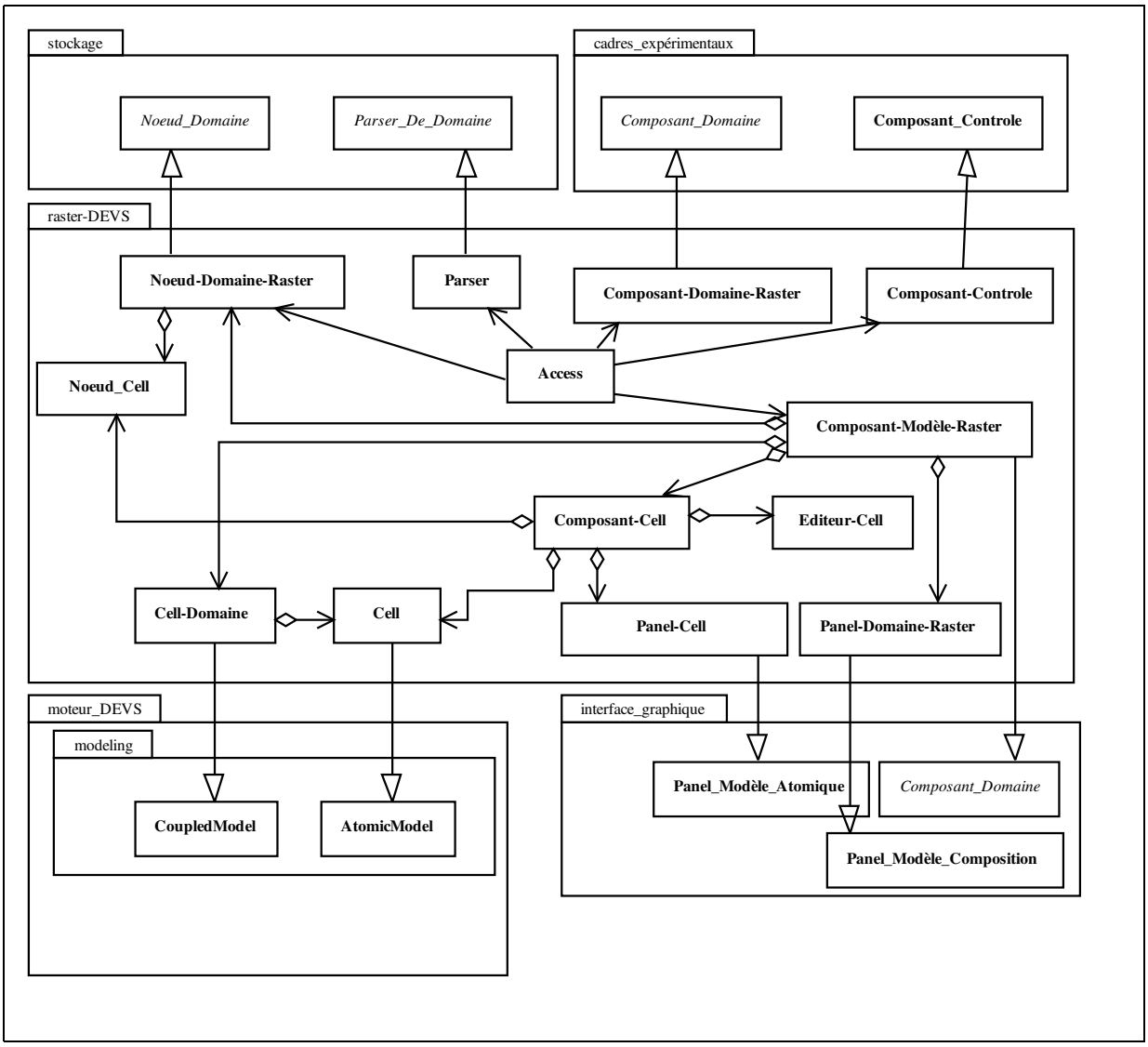

FIG. 4.7*: Le package raster-DEVS*

à l'intérieur du cadre expérimental sans pour autant charger un composant par objet **Cell** comme l'aurait exigé le recours à un parser de modèle diagramme.

La simulation de certains systèmes se répartissant sur de larges espaces et nécessitant de hautes résolutions, pose encore des problèmes en modélisation cellulaire. Ces problèmes peuvent être résolus par l'augmentation de la vitesse des ordinateurs, par une optimisation des techniques de simulation ou par une gestion de répartition de la charge de calcul. La section suivante propose une autre approche de ce problème, en introduisant la modélisation de propagation d'interface. Cette approche originale s'applique à l'étude de phénomènes physiques à interface : l'essentiel des observations concerne l'évolution de la frontière entre deux systèmes liés dans un même événement (par exemple : un front de flammes dans un événement incendie).

4.4 Modélisation de propagation d'interface

La technique d'étude de phénomènes de propagation d'interface existe en physique et est plus connue sous le nom anglais de "Front tracking" [Glimm et al., 1996], [Risebro et Holden, 2002], [Risebro et Tveito, 1992]. Cette technique s'intègre tout à fait dans le cadre de la simulation à événements discrets puisqu'elle permet de simuler l'enveloppe d'un phénomène en calculant ses changements structurels lorsque ces changements sont détectés et non pas à intervalles de temps réguliers. Cependant il n'existe pas de méthodologie DEVS permettant la spécification de modèles utilisant cette technique. Cette section présente les intérêts et cas d'utilisation de la méthode avant d'en présenter les spécifications comme modèle DEVS et le détail de son intégration dans le cadriciel.

4.4.1 Cas d'utilisation des modèles de propagation d'interface

Le principe général de la modélisation par propagation d'interface s'appuie sur le fait qu'un grand nombre de phénomènes physiques de propagation présentent des interfaces c'est-à-dire des zones frontières entre un phénomène et son milieu. Ainsi une crue, un tremblement de terre comme un feu, présentent des fronts de propagation comme interface entre le phénomène et le milieu dans

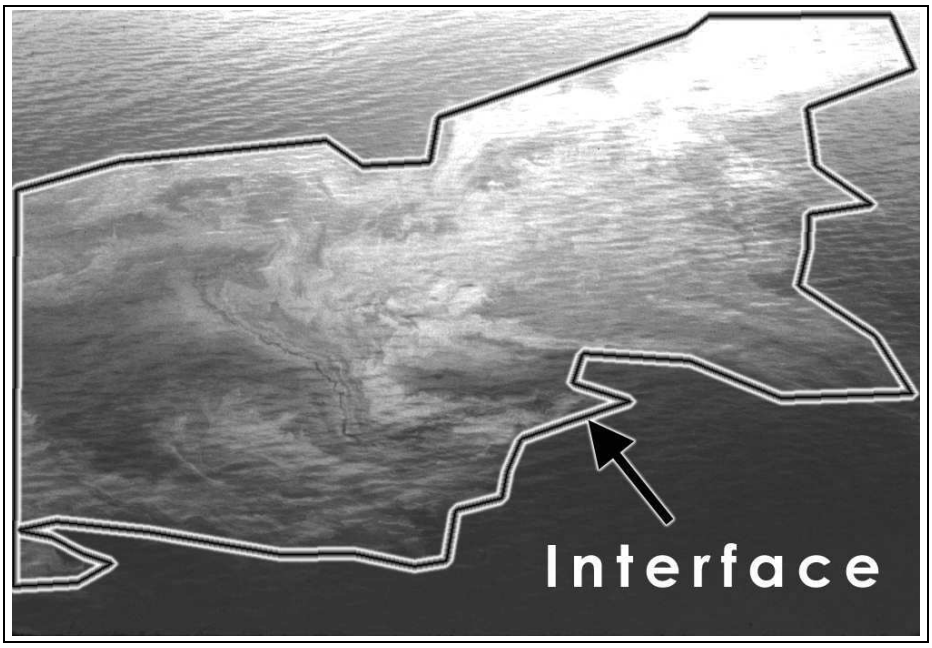

FIG. 4.8*: Photographie aérienne d'une nappe d'hydrocarbures montrant l'interface entre le phénomène et son milieu (la mer)*

lequel il se propage. Dans certains cadres expérimentaux, l'étude ne focalise que sur l'évolution de cette interface physique. En utilisant une approche cellulaire classique, la totalité de la carte doit être évaluée pour faire apparaître cette interface. En utilisant des interfaces vectorielles, seule l'évolution du contour du phénomène est modélisée puis simulée sur des cartes vectorielles et non matricielles. La figure 4.8 présente une nappe d'hydrocarbure où l'interface physique de la nappe avec la mer a été mise en évidence.

La simulation sur cartes vectorielles ne peut évidemment pas s'appliquer à l'étude de tous les phénomènes. Elle implique que le système à simuler présente un comportement constant et prévisible dans un espace homogène. Parmi ces phénomènes nous pouvons noter les propagations d'ondes de crues ou encore les modèles de propagation de feux basés sur l'équation de diffusion de la chaleur. Sur ces types de phénomènes, la simulation sur cartes vectorielles présente de sérieux avantages par rapport à la simulation sur cartes matricielles :

– L'espace ne doit pas être "discrétisé". De ce fait, ne se pose pas la question de la taille du maillage à employer. Les détails souvent négligés pour cause de taille trop importante du

domaine à étudier peuvent ainsi être pris en compte sans augmenter fortement la complexité (par exemple une route dans un modèle de propagation de feux de forêts).

- Les calculs redondants nécessaires lors de l'étude de la propagation d'un front stable dans un milieu homogène (plusieurs cellules semblables contiguës) peuvent être simplifiés car seul sera calculé le prochain événement correspondant au prochain changement de milieu.
- Une grande majorité de cartes étant disponibles directement au format vectoriel, l'étape de conversion vers des cartes matricielles est inutile et évite ainsi toute perte d'information.
- Les différents types de maillages induisent des déformations dans les propagations des modèles cellulaires si les différentes propriétés de l'espace (par exemple vent, pente, cotes) ne sont pas alignées avec la grille.

Du point de vue logiciel, une technique similaire est déjà utilisée dans des simulateurs spécifiques tels que FARSITE [Finney et Andrews, 1994] (pour la simulation de propagation de feux de forêts). Cet outil a obtenu une reconnaissance générale de la communauté (simulateur utilisé par l'US Department of Forest Management et par la majorité des bureaux d'études européens). Il est toutefois limité dans son domaine d'application à l'étude de la propagation des feux de forêts. FARSITE utilise exclusivement un principe modifié de la propagation des ondes de Huygens en temps discret pour déterminer l'avancement du phénomène.

En Physique, la méthode de "front tracking" n'est pas réservée à un certain type de problèmes, mais nécessite un cadre dans lequel les propriétés et la dynamique des interfaces doivent être définies [Dafermos, 1972]. Ainsi, la structure d'une interface est décrite par un ensemble de points, qui, une fois reliés, forment un polygone [Glimm et al., 1996]. Chacun de ces points possède un vecteur de déplacement, constitué d'une direction et d'une vitesse. L'interface évolue à l'intérieur d'une carte vectorielle en calculant les temps de chaque changement structurel de l'interface. Ces changements peuvent être déclenchés par trois types d'événements :

- Une collisions interne, intersection de certaines lignes du polygone décrivant l'interface, ce qui déclenche une recomposition.
- Une auto-décomposition, par le franchissement d'une distance limite. Lorsque le polygone

décrivant l'interface atteint une taille critique nécessitant l'ajout de points pour décrire sa forme. L'interface est alors raffinée.

– Une collisions externe, collision avec un élément de l'espace dans lequel l'interface évolue, ce qui déclenche une décomposition.

L'algorithme utilisé pour la résolution de modèles utilisant la technique de "front tracking" [Dafermos, 1972] est le suivant :

```
Faire les résolutions initiales des vitesses des fronts;
Calculer pour chacun des fronts le temps avant la collision;
T = temps à la première collision;
Tfin = temps décidé de fin de la simulation;

Tant que (T < Tfin)
Debut
      Si collision interne
           enlever les fronts en collision;
           calculer les nouvelles valeurs de progression des fronts;
           insérer les fronts resultant de suppressions;
      sinon si auto-decomposition
           calculer les fronts à décomposer;
           calculer les nouvelles valeurs de progression des fronts;
           insérer les fronts;
           calculer les nouvelles collisions possibles;
      sinon (collision externe)
           calculer les nouveaux fronts en fonction;
           - du type de frontière rencontrée;
           calculer les nouvelles collisions possibles;
      Fin si;
      Fin si;
      T = prochain temps de collision;
Fin
```

Algorithme 1*: Algorithme de "front tracking"*

La résolution de modèles de "front tracking" s'obtient à partir de codes spécifiques, développés en général pour la résolution d'un type particulier de problèmes [Glimm et al., 1996]. Cependant il est nécessaire de trouver une structure DEVS de description de tels modèles afin de permettre leur intégration dans l'environnement de modélisation proposé dans cette étude. La prochaine section

propose la méthodologie Vector-DEVS permettant la spécification des propriétés et de la dynamique des interfaces.

4.4.2 Vector-DEVS

Pour permettre la spécification de modèles basés sur les vecteurs, l'interface physique est décrite à l'aide d'agents géographiques ayant des coordonnées en deux dimensions qui peuvent bouger dans l'espace, en changeant leurs attributs spatiaux en fonction des milieux rencontrés. Un ensemble de ces agents représente une forme figurant un phénomène. Pour constituer une forme chaque agent possède un lien structurel vers le prochain agent, tous étant reliés dans le sens direct (sens inverse des aiguilles d'une montre).

4.4.3 Spécification des modèles Vector-DEVS

Le formalisme DSDEVS (Dynamic structure DEVS) [Barros, 1997] est utilisé dans le cas de modèles Vector-DEVS pour permettre de décrire de tels modèles sous une forme DEVS. DSVEVS permet la spécification de réseaux dynamiques structurés de modèles à événements discrets. L'utilisation du formalisme DSDEVS nous permet d'obtenir des modèles héritant des propriétés de DEVS ; la propriété de fermeture sous couplage de DSDEVS étant prouvée dans [Barros, 1997]. DSDEVS est ici utilisé comme le formalisme sur lequel est construit Vector-DEVS pour spécifier les méthodes spécifiques que requièrent la définition de modèles à interfaces dynamiques.

Un phénomène est décrit par sa forme qui évolue en structure dans l'espace et à travers le temps, s'adaptant aux conditions du milieu qu'il rencontre et dans lequel il se déplace. En utilisant cette technique, le comportement d'un phénomène est modélisé par la définition de l'agent qui représente un point quelconque du contour du phénomène. Les agents sont reliés entre eux dans le sens direct par des liens topologiques constituant ainsi une forme. Pour évoluer dans l'espace, chaque agent possède un vecteur de déplacement qui est utilisé pour calculer le temps avant chaque transition de milieu. La technique s'appuie sur deux entités chacune dérivée de DSDEVS pour décrire les systèmes : l'agent géographique, modèle DEVS basique et le gestionnaire de forme, réseau

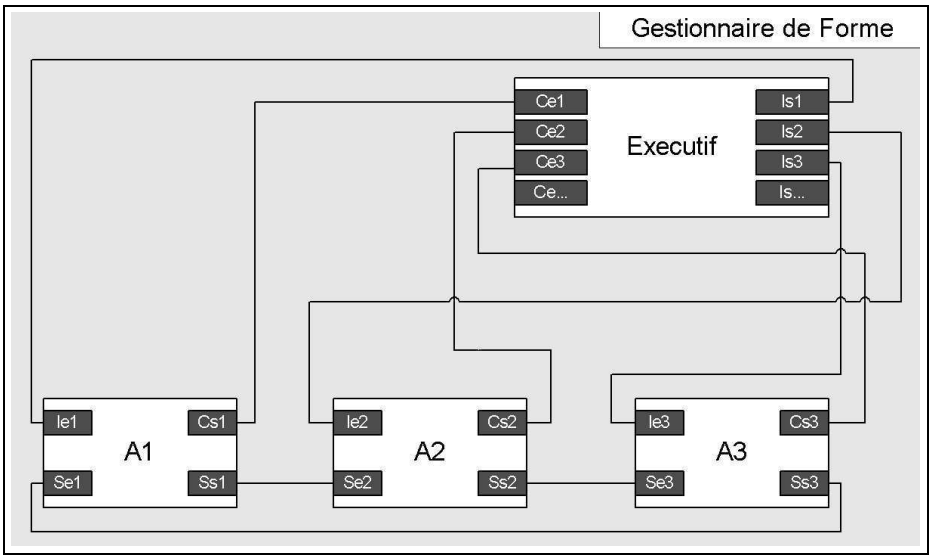

FIG. 4.9*: Vue structurelle des couplages à l'intérieur d'un gestionnaire de formes (en gris clair),* Exécutif *étant l'exécutif du gestionnaire, les modèles* A(1,2,3) *représentant les agents géographiques, les carrés gris foncés les ports des modèles et les lignes les couplages du gestionnaire.*

DSDEVS. La figure 4.9 présente la vue structurelle des couplages à l'intérieur d'un gestionnaire de formes, les modèles représentant les agents géographiques *A(1,2,3)* sont couplés ensemble par les ports *SE* et *SS* et reliés à l'exécutif du gestionnaire par les ports *Ie* et *Cs*. Ces différents composants possèdent les spécifications suivantes :

L'*Agent géographique*, *A*, est un modèle DEVS basique représentant un point du contour du phénomène. Il peut déclencher la génération de nouveaux agents ainsi qu'un changement de position. Il est décrit par :

$$A = < X(Ie,Se), S(Vd,Vn,TCs,E), Y(Cs,Ss), \delta_{ext}, \lambda, \delta_{int}, t_a > \text{ avec}$$

- $X(Ie,Se)$: Ensemble des entrées comprenant :
  - Ie : Port d'entrée recevant les informations sur l'ensemble des obstacles immédiats situés dans la zone où l'agent évolue. Les informations sont reçues sous la forme $< Ff\{Po(Ff)(x,y), Pd(Ff)(x,y)\} >$: ensemble des fronts fixes avec Po : coordonnées (x,y) du point d'origine et Pd : coordonnées (x,y) du point de destination.
  - Se : Port d'entrée recevant le vecteur de déplacement de l'agent lié topologiquement à A.

- $S(Vd, Vn, TCs, E)$: Ensemble des états comprenant :
  - Vd : Le vecteur de déplacement sous la forme $Vd(A) =< O(x,y), C, \Theta >$ avec
    - O : coordonnée d'origine (x,y),
    - C : vitesse, exprimée suivant le système de mesure choisi,
    - Θ : angle absolu de propagation en Radian dans l'espace, dans le sens direct.
  - Vn : Le vecteur de déplacement de l'agent lié à A sous la même forme $Vn(A) =< O(x,y), C, \Theta >$,
  - $TCs(Dec, ADec, Rec)$: Les types de changement de structure avec :
    - Dec : décomposition,
    - $ADec$: auto-décomposition,
    - Rec : recomposition.
  - E : Les autres propriétés locales de l'espace permettant de calculer Vd.
- Y : Ensemble des sorties comprenant :
  - Cs : Port de sortie informant de la prochaine décomposition de ce point,
  - Ss : Port de sortie informant l'agent lié topologiquement de son vecteur de déplacement.
- δ_{ext} : Fonction de transition externe avec :
  - $E \times Ie \rightarrow E$ Met à jour les propriétés de l'espace.
  - $Vn \times Vn \rightarrow S$ Prend en compte le nouvel état de l'agent connecté à A.
- λ : Fonction de sortie avec :
  - $TCs \rightarrow Ss$ Informe du type de changement de structure demandée, (décomposition, auto-décomposition, recomposition).
  - $Vd \rightarrow Cs$ Émet à l'agent suivant le vecteur de déplacement courant d'A.
- δ_{int} : Fonction de transition interne $S \times S \rightarrow S$, calcule notamment la valeur de Vd en fonction des nouvelles propriétés de l'environnement E.
- t_a : Renvoie le temps au prochain changement structurel en calculant les collisions avec les obstacles de l'environnement ou le temps à la prochaine auto-décomposition.

Le *Gestionnaire de Forme* est un réseau DSDEVS classique. C'est un conteneur pour tous les agents, en charge de leurs activation ainsi que des changements de structures. La structure est un

ensemble de connexions entre les agents, constituant ainsi une forme (polygone), cette structure est contenue dans l'*exécutif* du gestionnaire, χ.

Durant la phase d'initialisation, le *Gestionnaire de Forme* génère le nombre d'agents nécessaires pour composer la forme initiale du phénomène. Le *Gestionnaire de Forme GF* d'un réseau N a la structure suivante :

$$GF_N =< X_N, Y_N, \chi, M_\chi > \text{ avec}$$

- N le nom du réseau,
- X_N est l'ensemble des valeurs d'entrée du réseau,
- Y_N est l'ensemble des valeurs de sortie du réseau,
- χ est le nom de l'exécutif du réseau dynamique,
- M_χ est le modèle de l'exécutif χ.

Dans ce gestionnaire de forme, l'*exécutif du réseau* χ est un composant spécial qui possède la connaissance de la structure et de la dynamique de structure de la forme. M_χ, le modèle de l'exécutif est un modèle DEVS basique modifié.

L'exécutif lie tous les agents ensembles, la structure est stockée dans $\sum$. χ contient la fonction γ qui définit la dynamique de la structure de la forme : Par exemple, le principe de la propagation des ondes de Huygens peut être implémenté dans cette fonction pour décrire la propagation d'un front d'onde. γ provoque aussi les ajouts ou suppressions d'agents à la forme. M_χ est décrit par :

$$M_\chi =< X_\chi(Ce1..Cen), S_\chi(TCs, E), Y_\chi(Is1..Isn), \gamma, \sum^*, \delta_\chi, \lambda_\chi, ta_\chi >$$

ici,

- $X_\chi(Ce1..Cen)$: est l'ensemble des entrées de $Ce1$ à Cen (figure 4.9) recevant les informations concernant le type de changement de structure réalisable au temps courant,
- $S(TCs, E)$: est l'ensemble des états internes comprenant :
  - $TCs(Dec, ADec)$: Les règles de transition de structure (décomposition, auto-décomposition),
  - E : toutes les formes constituant l'environnement (carte vectorielle).

- $Y_\chi(Is1..Isn)$: est l'ensemble des sorties de $Is1$ à Isn informant l'agent sur l'environnement sous la forme $< Ff\{Po(Ff)(x,y), Pd(Ff)(x,y)\} >$ décrite précédemment.
- $\delta_\chi : \sum \rightarrow \sum$: est la fonction effectuant la recomposition de la structure résultant de calculs d'intersections interne,
- $\lambda_\chi : S_\chi \rightarrow Y_\chi$: fonction de sortie sélectionnant les frontières de l'environnement pouvant faire l'objet de collisions avec le vecteur auquel la sortie est destinée.
- $t_a\chi$: Renvoie le temps d'occurrence du prochain changement structurel résultant de recomposition, ou 0 si un changement est demandé par un agent.
- $\gamma : S_\chi \rightarrow \sum^*$: est la fonction de changement de structure,
- $\sum^* \times TCs \rightarrow \sum^*$: est l'ensemble des structures pouvant résulter de l'application des règles de transition de structures dans le modèle.

Une structure $\sum_\alpha \in \sum^*$ et ses états partiels associés $S_{\alpha,\chi} \in S_\chi$, sont donnés par

$$\sum_\alpha = \gamma(s_{\alpha,\chi}) = (D_\alpha, \{M_{i,\alpha}\}, \{I_{i,\alpha}\}, \{Z_{i,\alpha}\}) \text{ où}$$

- D_α : est l'ensemble des composants associés avec $s_{\alpha,\chi}$,
- pour tout $i \in D_\alpha$, $M_{i,\alpha}$ est le modèle DEVS classique du composant i,
- pour tout $i \in D_\alpha \bigcup \{\chi, N\}$,$\{I_{i,\alpha}\}$ est l'ensemble des composants influençant i,
- pour tout $i \in D_\alpha \bigcup \{\chi\}$, $\{Z_{i,\alpha}\}$ est la fonction d'entrée du composant i et $\{Z_{N,\alpha}\}$ est la fonction de sortie du réseau.

Cette section a présenté les spécifications formelles de modèles Vector-DEVS en DSDEVS qui nous a rendu possible la formalisation de modèles de type "front tracking". Le modélisateur désireux d'utiliser cette technique n'a qu'un nombre limité de fonctions à spécifier. Ces fonctions sont illustrées dans la section suivante par l'exemple d'un type de propagation d'interface utilisant le principe de Huygens [Glimm et al., 1996] pour la décomposition d'interfaces.

4.4.4 Fonctions spécifiques des modèles Vector-DEVS

Cette section présente graphiquement et illustre par un exemple, les fonctions spécifiques devant être implémentées pour représenter un modèle Vector-DEVS. Le modèle de "Front tracking"

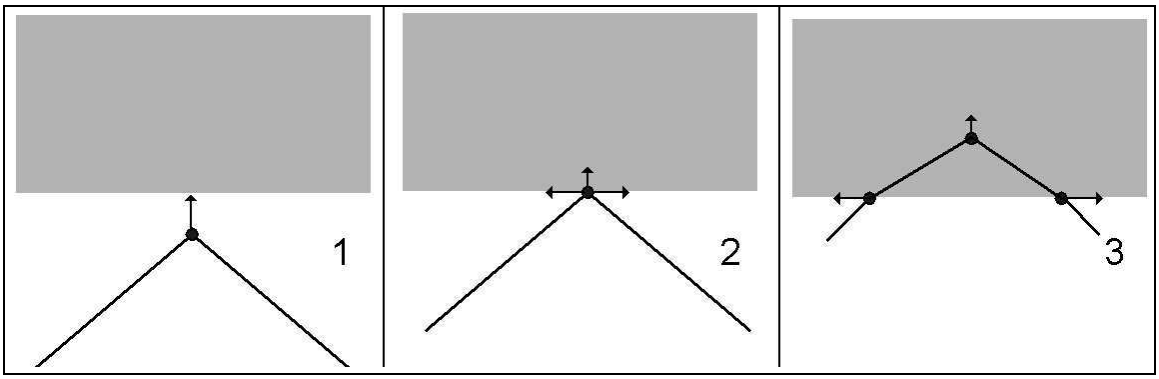

FIG. 4.10*: Décomposition selon le principe de Huygens avec un agent entrant en collision avec un milieu moins rapide, (1) avant la collision, (2) l'agent se décompose en trois, (3) après la décomposition.*

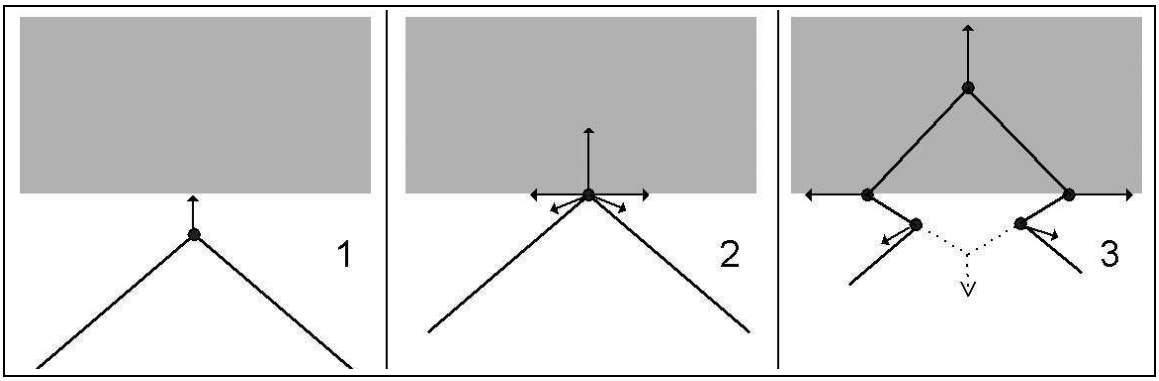

FIG. 4.11*: Décomposition selon le principe de Huygens avec un agent entrant en collision avec un milieu plus rapide, (1) avant la collision, (2) l'agent se décompose en cinq, (3) après la décomposition.*

servant d'exemple est tiré de [Glimm et al., 1996] et est basé sur le modèle de propagation des ondes de Huygens. Le lecteur intéressé trouvera dans cet ouvrage les solution mathématiques des problèmes physiques associés. Pour décrire un modèle Vector-DEVS, le modélisateur doit définir huit fonctions :

- la fonction de décomposition *Dec*. Cette fonction est appelée à l'intérieur de la fonction γ de χ. Dans le cas de modèles de propagation d'ondes, cette fonction peut générer quatre comportements différents résultat d'une décomposition après une collision :
  - d'un point d'un milieu rapide sur une ligne d'un milieu plus lent (Figure 4.10), ici l'agent

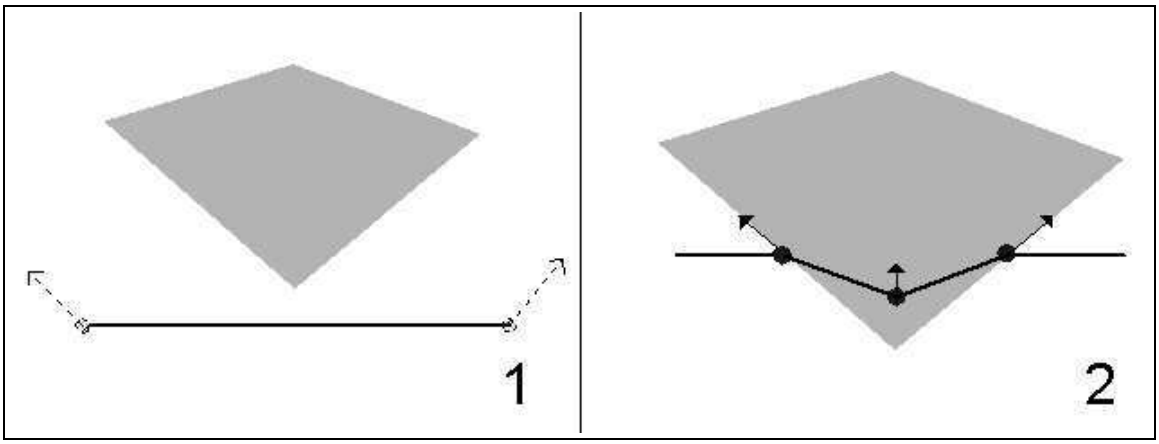

FIG. 4.12*: Décomposition selon le principe de Huygens avec un lien entre agents entrant en collision avec un milieu moins rapide, (1) avant la collision, (2) après la décomposition en trois agents.*

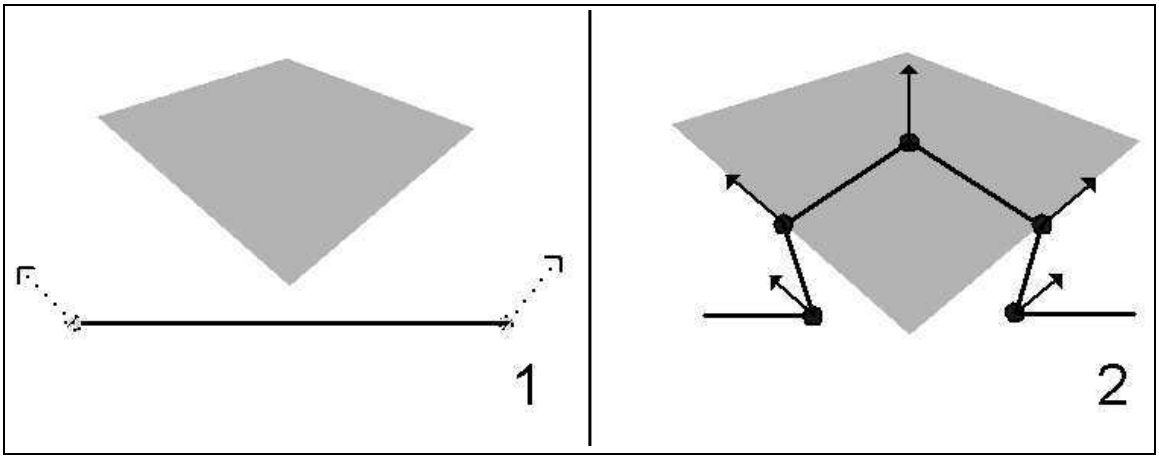

FIG. 4.13*: Décomposition selon le principe de Huygens avec un lien entre agents entrant en collision avec un milieu plus rapide, (1) avant la collision, (2) après la décomposition en cinq agents.*

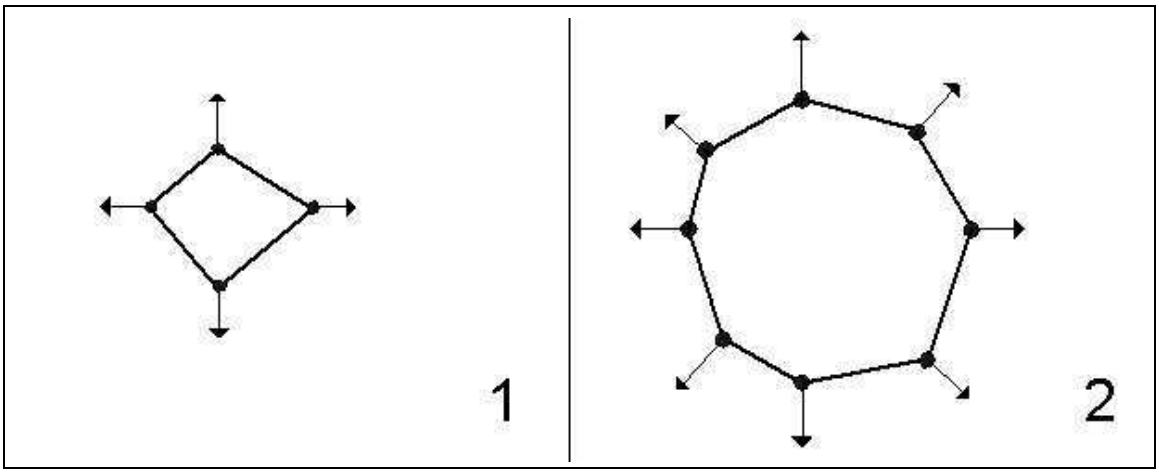

FIG. 4.14*: Auto-décomposition (raffinement) après franchissement de distance limite, (1) avant la décomposition, (2) forme "raffinée".*

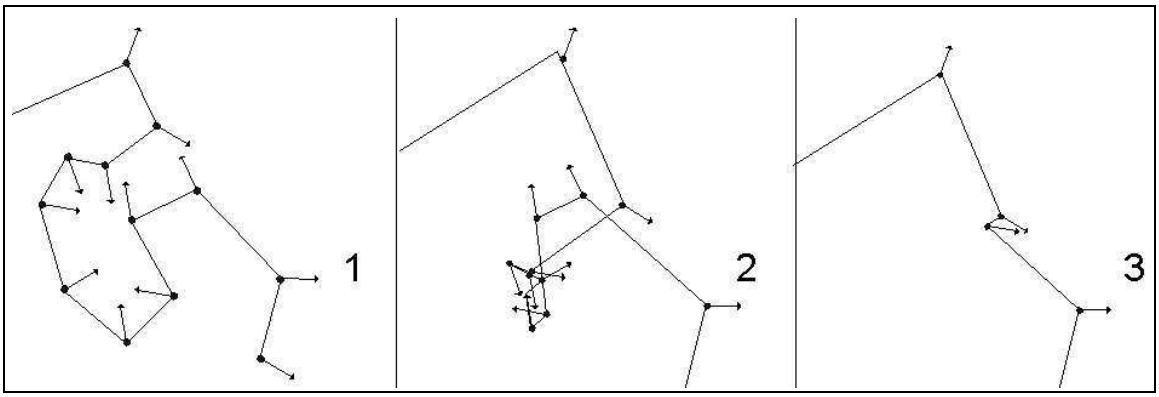

FIG. 4.15*: Recomposition après détections d'intersection, (1) avant les intersections, (2) intersections détectées, (3) après la recomposition.*

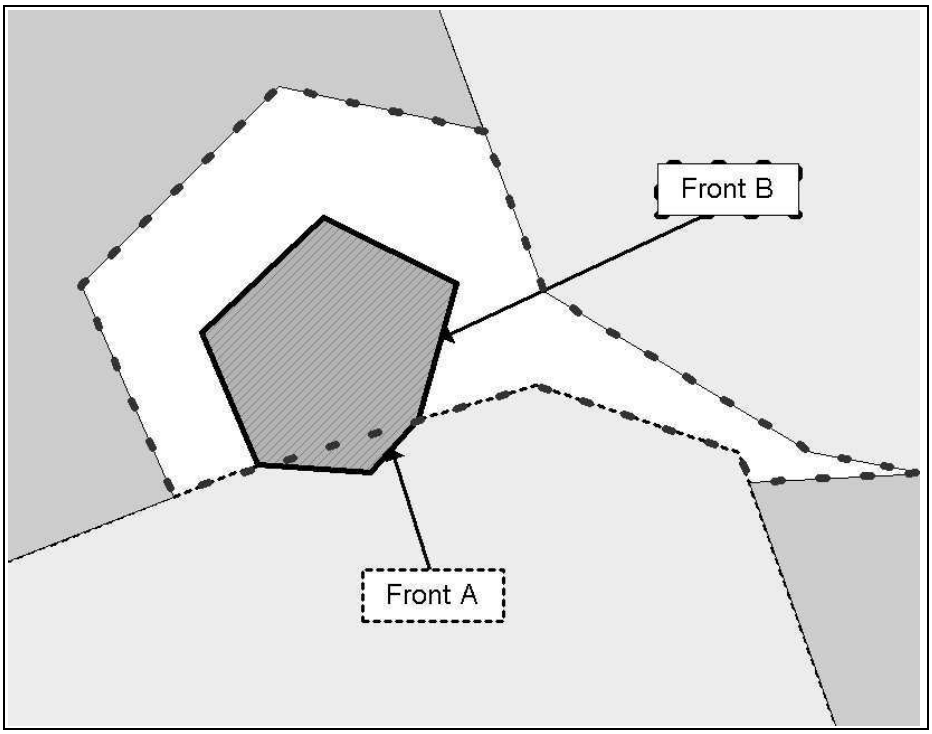

FIG. 4.16*: Sélection des frontières de l'environnement pouvant faire l'objet de collision avec un vecteur : les frontières pour le front A (et ses deux agents associés) sont signalées par un trait pointillé fin et les frontières pour le front B (et ses deux agents associés) sont signalées par un trait pointillé gras.*

se décompose en trois : deux des agents suivent la frontière à la vitesse la plus rapide et le troisième perpendiculairement à l'interface.

– d'un point d'un milieu lent sur une ligne d'un milieu plus rapide (Figure 4.11), ici l'agent se décompose en cinq, deux des agents suivant la frontière à la vitesse la plus rapide et deux autres agents étant créés pour prendre en compte l'effet de réflexion de la frontière avec un milieu plus rapide.

– d'une ligne d'un milieu rapide sur un point d'un milieu plus lent (Figure 4.12), ici, comme dans le cas d'une collision avec une ligne, l'agent se décompose en trois, deux des agents suivant la frontière à la vitesse la plus rapide.

– d'une ligne d'un milieu lent sur un point d'un milieu plus rapide (Figure 4.13), ici, comme dans le cas d'une collision avec une ligne, l'agent se décompose en cinq, deux des agents suivant la frontière à la vitesse la plus rapide et deux autres agents sont créés pour prendre en compte l'effet de reflection de la frontière avec un milieu plus rapide.

- la fonction d'auto-décomposition *ADec*. Elle est aussi appelée par la fonction γ de χ lorsque l'agent demande un tel type de décomposition. La figure 4.14 montre une auto-décomposition de quatre agents en huit, après qu'un ou plusieurs agents aient fait la requête d'une décomposition. Ici l'approximation faite par le polygone les reliant est trop grossière par rapport à la forme idéale voulue (ici un ovale).
- La fonction de recomposition *Rec*. Elle est appelée par la fonction δ de χ lorsque l'exécutif détecte des intersections à l'intérieur de la forme qu'il dirige. La figure 4.15 montre la recomposition d'une structure ; les agents ayant des coordonnées comprises à l'intérieur de l'enveloppe externe de la forme sont supprimés et de nouveaux agents recréés aux points d'intersection.
- La fonction de sélection de voisinage. Elle est appelée par la fonction λ de χ. Cette fonction teste l'appartenance d'un agent aux polygones de l'espace où l'agent se trouve. La figure 4.16 montre la sélection de frontières pour les deux agents composant le front A (en pointillés fin) et le front B (en pointillés gras). Ces informations sont retransmises aux agents concernés pour les calculs de détection de collision.
- La fonction de détection de collision. Elle est appelée par la fonction t_a de l'agent pour déterminer le temps jusqu'à la prochaine collision.
- La fonction de détection d'intersection. Elle est appelée par la fonction t_a de χ pour déterminer le temps jusqu'à la prochaine recomposition résultant d'intersections.
- La fonction de détection d'auto décomposition. Elle est appelée par la fonction t_a de l'agent pour déterminer le temps jusqu'à la prochaine auto-décomposition ayant lieu lorsque l'agent a effectué une certaine distance sans collision.
- La fonction de changement de vecteur de déplacement. Elle est appelée par δ_{int} de l'agent pour déterminer les nouvelles valeurs de son vecteur de déplacement en fonction des nouvelles propriétés de l'espace qui lui ont été données par χ. Dans le cas du modèle de propagation des ondes de Huygens, le vecteur est calculé pour que le front constitué entre l'agent et son suivant garde un angle constant (avance sur un axe perpendiculaire à la droite passant par les deux agents).

La simulation de modèles Vector-DEVS s'effectue grâce aux simulateurs abstraits de modèles DSDEVS décrits par Barros dans [Barros, 1998]. La section suivante introduit ces simulateurs et comment ils ont été adaptés dans le cadre de Vector-DEVS.

4.4.4.1 Simulateurs de modèles Vector-DEVS

Les modèles Vector-DEVS sont décrits à l'aide de DSDEVS et peuvent donc être simulés grâce aux simulateurs abstraits définis par Barros dans [Barros, 1998]. Barros définit trois entités (*synchronisateur*, *simulateur de réseau* et *simulateur*) et trois types de messages (*START*, *TRANSITION* et *OUTPUT*) pour la simulation de modèles DSDEVS. Deux autres messages sont aussi définis, le premier, *ERREUR*, pour permettre de détecter des problèmes, le second, *RETURN*, pour signaler une fin de fonction sans résultat.

Le *synchronisateur* coordonne toute la simulation, il est attaché au *simulateur de réseau* qui se situe au niveau le plus haut de l'arbre de simulation. Lorsqu'il reçoit le message *START* le *synchronisateur* est chargé d'initialiser les modèles par l'envoi du message *START* à tous les sous-modèles (*C*) puis il déclenche les sorties des sous-modèles par l'envoi du message *OUTPUT* et enfin les transitions par le message *TRANSITION* successivement jusqu'à la fin de la simulation. Par la suite, nous utiliserons les abréviations suivantes : `T` représente le temps courant, `Tl` le temps d'occurrence de l'événement précédent et `Tn` le temps à l'événement suivant. L'algorithme de simulation du *synchronisateur* est le suivant :

```
Quand reception(START,T)
Debut
- Tant qu'un composant à une transition de prévue
   Tantque(Tn(C) ≠ ∞)

- Tn = temps au prochain événement, C = index des composants
      Debut
- Tc, temps courant, prends la valeur du temps
- de la transition la plus immédiate
          Tc = min(Tn(C))
- Envoie le message OUTPUT aux composants devant s'activer à Tc
          Envoi(OUTPUT,Tn,C)
- Envoie le message TRANSITION aux composants devant s'activer à Tc
          Envoi(TRANSITION,Tn,C)
      Fin
   Fin
```

Algorithme 2*: Algorithme d'initialisation du* synchronisateur

Un *simulateur de réseau* est attaché à chaque *Réseau DSDEVS* contenant les modèles DEVS basiques et l'exécutif de ce réseau. L'ensemble des modèles est noté C, ce même ensemble sans l'exécutif χ est noté D. L'algorithme d'initialisation du *simulateur de réseau* est le suivant :

```
Quand reception(START,T)
Debut
- Envoie le message START a tous les composants de C
      Envoi(START,T) à {I|I ∈ C}
      Tl ← T
      Tn ← min(Tn(C))
Fin
```

Algorithme 3*: Algorithme d'initialisation du* simulateur de réseau

Dans le cas de Vector-DEVS, lors de l'initialisation les agents vont calculer leurs vecteurs de progression en fonction du milieu dans lequel ils viennent d'être créés. Une fois les modèles initialisés la simulation peut s'effectuer. L'algorithme régissant la réaction du *simulateur de réseau* aux transitions est le suivant :

```
Quand reception(TRANSITION,T,χ)
Debut
- Vérifie la validité du message
  Si T ∉ [Tl,Tn] alors ERREUR Fin Si
  Si T < Tn et χ = ∅ alors RETURN Fin Si
- Stocke l'ensemble des sous-modèles courants dans D' (ancien état)
  D' ← D
- Envoie TRANSITION à tous les sous modèles connectés à des entrées sauf χ
  Envoi(TRANSITION,T,Z_I(×_{j∈I_I} v_j)) à {I|I ∈ D}
- Envoie TRANSITION à χ
  Envoi(TRANSITION,T,Z_χ(×_{j∈I_χ} v_j)) à χ
- Envoie START aux composants créés après un changement de structure
  Envoi(START,T) à {I|I ∈ D − D'}
     Tl ← T
     Tn ← min{Tn_I|I ∈ C}
Fin
```

Algorithme 4*: Algorithme de réaction à la transition du* simulateur de réseau

Le *simulateur de réseau* déclenche les transitions de tous les modèles dans D avant de déclencher les transitions de χ pour que les changements de structures n'affectent que les transitions se déroulant après ce changement. Les modèles créés après le changement de structure sont initialisés par l'envoi du message *START*. Les sorties sont générées immédiatement avant la transition, tel que spécifié dans le formalisme DEVS, pour assurer que la sortie d'un modèle ne dépend pas d'une transition en cours. L'algorithme de réaction au message de sortie *OUTPUT* du *simulateur de réseau* est le suivant :

```
Quand reception(OUTPUT,T)
Debut
  Si T = Tn alors
     Envoie(OUTPUT,T) à {I|I ∈ C}
- Génère la sortie de tous les composants ayant des connexions de sorties
actives
     y ← Z_N(×_{I∈I_N} Y_I)
  Sinon
     y ← ∅
  Fin Si
Fin
```

Algorithme 5*: Algorithme de réaction au message de sortie du* simulateur de réseau

Dans Vector-DEVS les agents sont modélisés par des modèles DEVS basiques. Ces modèles sont simulés grâce aux *simulateurs* associés. Le *simulateur* réagit au message d'initalisation *START* par l'algorithme suivant :

```
Quand reception(START,T)
Debut
     Tl ← T
     s ← s_0
     Tn ← T + Ta(s)
Fin
```

Algorithme 6*: Algorithme d'initialisation du* simulateur

L'agent prend d'abord son état initial avant de fixer son temps au prochain changement en programmant ses changements structurels. Le *simulateur* est aussi chargé de la simulation de χ. Dans ce cas, le temps au prochain changement est soit celui de la recomposition de la structure, soit 0 dans le cas d'une requête de changement de structure immédiate par un agent.

Le *simulateur* réagit au message de transition *TRANSITION* par l'algorithme suivant :

```
Quand reception(TRANSITION,T,χ)
Debut
- Vérifie la validité du message
  Si T ∉ [Tl,Tn] alors ERREUR Fin Si
  Si T < Tn et χ = ∅ alors RETURN Fin Si
s ← δ(s,T − Tl,χ)
Tn ← T + Ta(s)
Fin
```

Algorithme 7*: Algorithme de réaction à la transition du* simulateur

Après la vérification de validité du message, le simulateur met à jour les états du modèle par l'appel à la fonction de transition δ puis programme les prochains temps d'activation du modèle.

Enfin, le comportement en sortie d'un simulateur est donné par l'algorithme suivant :

```
Quand reception(OUTPUT,T)
Debut
  Si T = Tn alors
     y ← λ(s)
  Sinon
     y ← ∅
  Fin Si
Fin
```

Algorithme 8*: Algorithme de réaction au message de sortie du* simulateur

La réception du message *OUTPUT* active le modèle immédiatement avant la transition. La fonction λ est alors appelée pour générer les sorties à partir des états courants.

Nous venons de presenter les simulateurs abstraits de modèles DSDEVS et leurs applications dans le cadre de Vector-DEVS. La section suivante présente rapidement une simulation de modèles à propagation d'interface pour illustrer ces algorithmes.

4.4.4.2 Exemple de simulation basée sur les vecteurs

La Figure 4.17 montre un modèle basé sur les vecteurs. Dans son état initial (1), le modèle est composé dequatre agents (A,B,C,D). Chaque agent possède aussi un "lien topologique" au

prochain agent et connaît sa position (A est lié à B, B à C, C à D et D à A). A chaque agent est attaché un vecteur de déplacement qui peut varier en vitesse et direction. Un agent se décompose lorsqu'il entre en collision avec une surface ou lorsqu'un des liens aux agents suivants entre en collision avec une nouvelle surface. La figure 4.17(1,2,3) illustre le processus de simulation d'un modèle vecteur simple. La figure 4.18 présente le diagramme de séquence des objets intervenant dans la simulation.

A l'état initial les agents sont instanciés pour représenter la forme originale du phénomène. Les vecteurs de déplacement sont ensuite calculés à partir des informations sur le milieu, envoyés par l'exécutif du gestionnaire de forme associé (figure 4.17(1)).

Le temps au prochain événement est ensuite calculé pour chacun des agents (fonction t_a) : On calcule d'abord l'équation du segment formé entre l'agent courant et son suivant à partir de son vecteur propre et de celui de l'agent auquel il est topologiquement attaché. Puis on calcule successivement les temps de collision entre ce segment et les segments constituant le polygone dans lequel évolue l'agent. Le temps suivant est alors celui de la collision arrivant la plus tôt.

Une fois que chaque agent a calculé sa prochaine collision, la forme est placée dans son état suivant (figure 4.17(2)), puis le gestionnaire de forme met à jour les propriétés de chaque agent en fonction des nouvelles propriétés de l'espace fournies par l'exécutif du gestionnaire de forme. Dans la figure 4.17(2), le point A fait une requête immédiate (au temps $T+0$) de changement de structure auprès de l'exécutif du gestionnaire de forme suite à une collision. Ce changement de structure s'effectue après les transitions des agents, assurant qu'aucun changement structurel ne puisse s'effectuer avant que toutes les transitions des agents aient été effectuées.

L'agent A génère l'instantiation de deux autres agents par une requête de changement de structure ; ces deux agents mettent alors à jour leur comportement en fonction des propriétés de l'espace qui leur est fourni par le gestionnaire spatial. Les agents de la forme calculent ensuite le temps au prochain événement et transitent vers l'état de la figure 4.17(3).

Cette section a illustré la simulation d'un modèle Vector-DEVS sur un carte vectorielle. Vector-DEVS respecte les spécifications de DSDEVS et en conséquence hérite des propriétés de DEVS. Un modèle Vector-DEVS peut donc être intégré au même titre que les autres techniques de manière

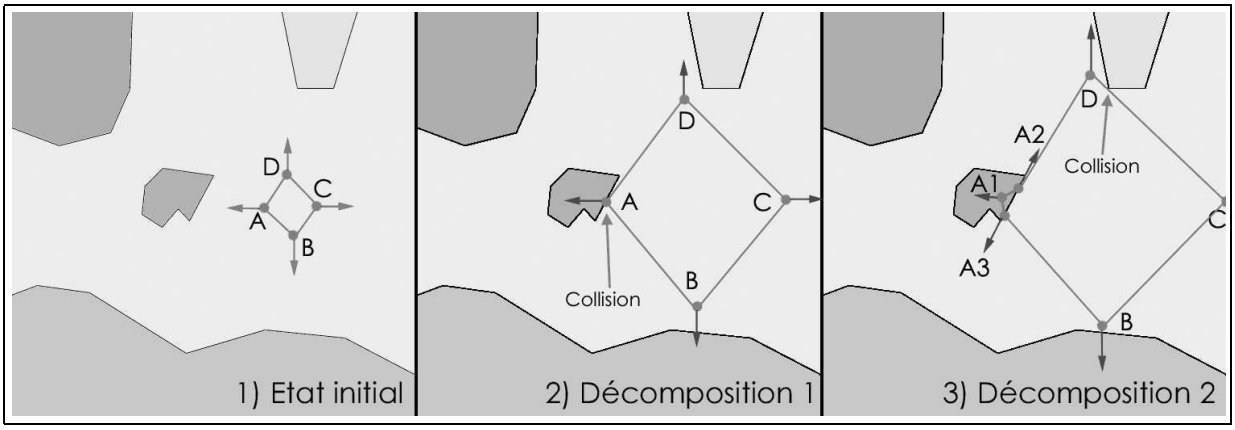

FIG. 4.17*: Exemple de propagation : (1) Etat initial, (2) première collision (3) Seconde collision*

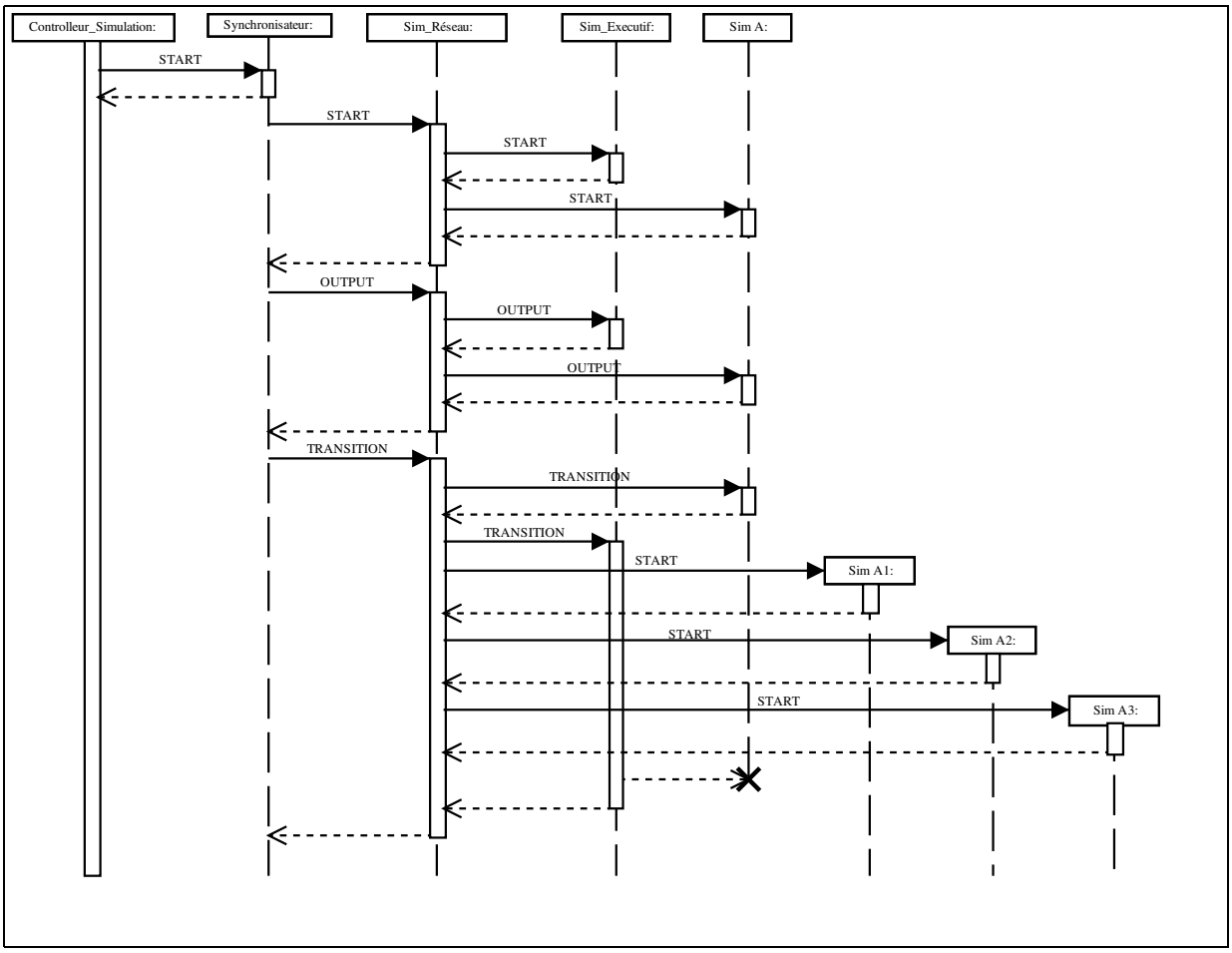

FIG. 4.18*: Diagramme de séquence de l'exemple de simulation*

modulaire et hiérarchique en respectant l'intégrité du modèle. L'intégration de Vector-DEVS dans l'environnement logiciel est traitée dans la section suivante.

4.4.4.3 Intégration de Vector-DEVS dans le cadriciel

Plusieurs problèmes se posent du point de vue logiciel et en particulier, dans la méthodologie : la connaissance de tout l'espace dans lequel évolue un phénomène est contenue dans l'exécutif du gestionnaire de forme. Pratiquement cela induit des duplications de données géographiques en mémoire dans le cas où plusieurs formes évoluent dans un même espace. Cela impose aussi des procédures lourdes à implémenter de recherches spatiales et d'entrée sortie de données géographiques, pour un objet : l'exécutif n'a pas cette vocation. Pour éviter ces problèmes les contraintes spatiales sont gérées par le *gestionnaire spatial* dans l'environnement.

Dans le cas de Vector DEVS, le *Gestionnaire spatial* est lié à chaque gestionnaire de formes et possède la connaissance de la totalité de l'espace sur lequel se déroule la simulation. Pratiquement, il offre à l'exécutif du *gestionnaire de forme* un conteneur pour tous les états concernant l'environnement ainsi qu'un ensemble de méthodes pour manipuler ces informations.

Le *gestionnaire spatial* est une entité ayant la connaissance de la totalité des propriétés spatiales du cadre de la simulation. Il fournit aux modèles les informations permettant de mettre à jour leurs propriétés spatiales. Pour cela il doit travailler sur les types de représentation de l'espace matriciel et vectoriel, et fournir des translations éventuelles entre ces différents types de représentation (rasterisation et vectorisation).

Le *gestionnaire spatial* agit comme un proxy du SIG. La totalité de l'espace dans lequel se déroule la simulation est chargé au début de la simulation. Pendant la simulation les sorties ayant des répercussions sur cet espace sont stockés sous forme d'événements par le *gestionnaire spatial*. Ces données sont ensuite renvoyées au SIG à la fin de la simulation.

L'intégration de vector-DEVS dans le cadriciel est faite grâce au package *vector-DEVS* présenté en figure 4.19. Ce package contient les classes spécifiques aux techniques de modélisation par propagation d'interfaces où chaque agent est un modèle atomique. Dans l'interface graphique de modélisation, chaque domaine que l'on veut représenter par un ensemble de points est repré-

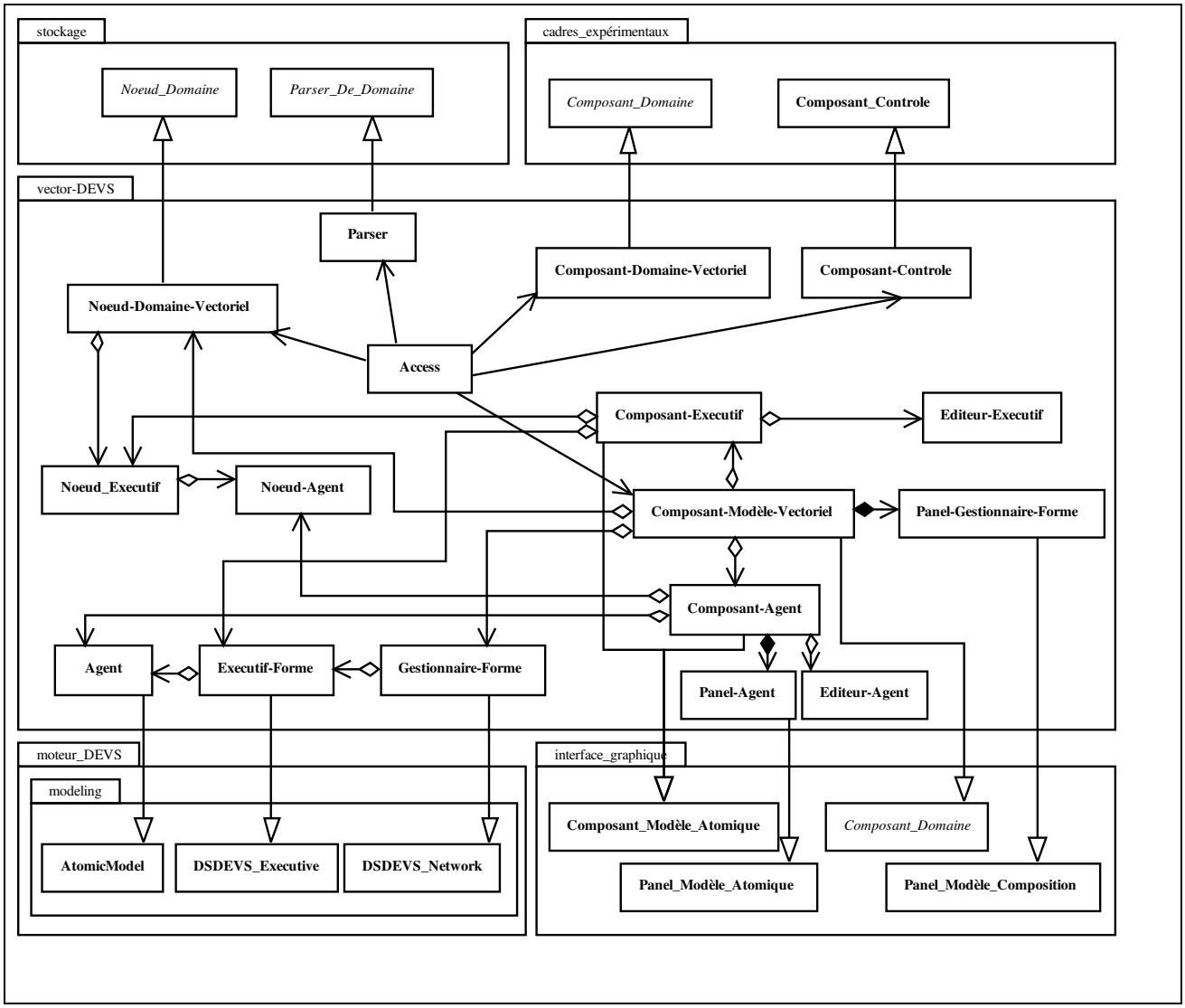

FIG. 4.19*: Le package Vector-DEVS*

senté par un objet **Composant-Modèle-Vectoriel** contenant les agents de type **Composant-Agent** possédant un éditeur de code pour l'objet de type **Agent**, héritant de **AtomicModel** et définissant son comportement. Chaque objet **Composant-Modèle-Vectoriel** est aussi associé à un objet de type **Composant-Executif** et son éditeur, pour modifier le code de l'objet **Executif-Forme** associé, qui est chargé de la dynamique du modèle.

Les objets **Agent**, **Executif-Forme** et **Gestionnaire-Forme** peuvent être simulés par de objets **Simulator** et **DSDEVS-Coordinator** standard et ne nécessitent donc pas la création spécifique de classes à cet effet. Les modèles vector-DEVS sont stockés par des objets de type **Noeud-Domaine-Vectoriel**, **Noeud-Executif** et **Noeud-Agent** et accédés grâce à un objet **Parser** spécifique héritant de ***Parser-De-Domaine***. Comme pour le package *raster-DEVS*, le **Parser** est spécifique car il doit réaliser le chargement d'un **Composant-Domaine-Vectoriel** qui affiche les états des objets **Agent** à l'intérieur du cadre expérimental sans pour autant charger un composant par objet **Cell** comme cela aurait été le cas avec un parser de modèle diagramme.

4.5 Conclusion

Ce chapitre a présenté les différents ajouts réalisés au cadriciel pour permettre l'intégration de différentes techniques de modélisation et de simulation. Il existe une grande quantité de paradigmes de modélisation, chacun étant adapté à un problème donné. A chaque paradigme est associé une technique de modélisation et de simulation. Ainsi, plus un modélisateur a de technique à disposition, plus il aura de solutions à ses problèmes. Nous avons voulu dans cette perspective, ajouter trois techniques adéquates au cadriciel afin de traiter des problèmes environnementaux.

La première, feedback-DEVS, est utile à la spécification de modèles adaptatifs, auto-apprenants, autorisant la construction de modèles empiriques. L'intégration de cette technique a été facilitée par sa similarité avec la technique DEVS classique déjà présente dans le cadriciel.

La seconde technique basée sur les automates cellulaires, est la méthode la plus fréquemment utilisée pour l'étude de systèmes ayant des dimensions spatiales. Pour rendre cette technique dis-

ponible, nous avons du ajouter des modes de représentation de l'espace à notre outils et développer les cadres expérimentaux et les interfaces vers des outils d'analyse spatiales (SIG).

Enfin, la troisième technique ajoutée, Vector-DEVS est une approche originale adaptée des méthodes mathématiques de "front tracking" [Glimm et al., 1996] permettant l'étude de la propagation d'un phénomène par la simulation de la dynamique de son interface physique. Pour intégrer cette approche nous avons cherché à appliquer le formalisme DSDEVS [Barros, 1997] pour qu'il prenne en compte la gestion de l'espace. Ce formalisme a été choisi car il autorise la définition de modèles à structure dynamique. Les coté spatial a été introduit par la définition des concepts d'*agent géographique* (modèle atomique géo-référencé) et de *gestionnaire de forme*(réseau DSDEVS intégrant une dynamique structurelle et spatiale). La spécification formelle Vector-DEVS a été présentée et son ajout dans le cadriciel effectué d'un point de conception orientée objet. L'intégration logicielle de cette technique est en cours de réalisation et constitue une des principales perspectives de recherches.

Pour chacune des trois techniques ajoutées, nous avons précisé les classes et packages obtenus à l'issue de leur intégration.

Le prochain chapitre est consacré à la réalisation logicielle du cadriciel, JDEVS.

CHAPITRE 5

Implémentation

JDEVS est une implémentation du cadriciel décrit dans le chapitre 3. Plusieurs raisons ont motivé le développement effectif de l'architecture générique ainsi que des techniques de modélisation présentées dans le chapitre 4. La première concerne la validation des choix d'architecture logicielle, certains problèmes ne devenant apparents que lors de l'implémentation de l'outil. Ainsi, l'expérience de ce développement nous a conduit d'abord à redéfinir l'architecture de classes du cadriciel afin qu'il puisse mieux correspondre aux attentes des utilisateurs. Nous avons ensuite publié le logiciel sur plusieurs sites web afin de faire valider les choix logiciels par la communauté la plus large possible. Le deuxième objectif poursuivi lors de ce développement à été de satisfaire le besoin d'un outil de modélisation convivial dans notre laboratoire. JDEVS est largement utilisé dans plusieurs projets de recherche ; trois des modèles développés sont d'ailleurs présentés dans le chapitre 6.

Ce chapitre est décomposé en quatre sections, la première présente les différents modules de JDEVS. Les techniques de modélisation et de simulation intégrées dans le logiciel sont ensuite détaillées dans les trois dernières sections.

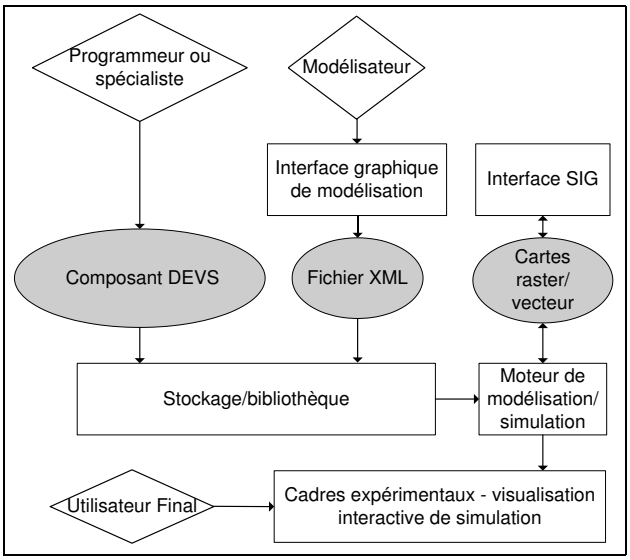

FIG. 5.1*: Vue des modules du logiciel JDEVS, les carrés correspondent aux modules, les losanges correspondent aux acteurs et les cercles aux formats d'échange de données*

5.1 Le logiciel JDEVS

JDEVS est composé de cinq modules indépendants présentés en figure 5.1 : un moteur de simulation, une interface graphique de modélisation, un module de stockage dans des bibliothèques, un module de connexions au SIG et les cadres expérimentaux de visualisation et de simulation. Chacun de ces modules reprend l'architecture de classe définie dans le cadriciel ce qui permet à ces modules de fonctionner ensemble et de pouvoir être éventuellement remis à jour indépendamment. Sans la prise en compte de techniques de modélisation spécifique, JDEVS prend pour base la technique de modélisation classic-DEVS [Zeigler, 2000]. La première section est consacrée au moteur de modélisation et de simulation implémentant cette technique.

5.1.1 Moteur de modélisation et de simulation

Le moteur de modélisation et de simulation est une implémentation de la méthodologie classic-DEVS avec ports. Dans l'attente d'une standardisation par le DEVS standardisation group [DEVS-Group, 2003], les modèles DEVS atomiques doivent pour l'instant être transformé en instructions JAVA .

```
public class DevsAtom extends BasicModel {
      Port i1 = new Port(this,"i1","IN");
      Port o1 = new Port(this,"o1","OUT");

public DevsAtom () {
      super("DevsAtom");
      states.setProperty("A","");  }
    EventVector outFunction(Message m) {
         return new EventVector();}
    void intTransition() {}
    EventVector extTransition(Message m) {
         return new EventVector();}
    int advanceTime(){return A;}
}
```

FIG. 5.2*: Code source JAVA pour un composant atomique DEVS standard*

Le formalisme DEVS propose des interfaces bien définies pour la description de systèmes. Ces interfaces nous permettent de pouvoir utiliser des modèles programmés dans de nombreux langages orientés objets et d'y accéder ensuite grâce à des appels de méthodes à distance (comme Java RMI). La modélisation de modèles atomiques peut toutefois se faire directement dans l'interface et en utilisant Java. Aussi, il est possible d'ajouter un composant atomique vide, de définir ses interfaces d'entrée/ sortie et de générer le squelette du code du modèle. Ce modèle vide est ensuite enregistré dans la bibliothèque et compilé. La figure 5.2 présente le code généré automatiquement pour

$DevsAtom = < X(i1), S(A), Y(o1), \delta_{int}, \delta_{ext}, \lambda, t_a >$

Les fonctions de sortie et de transition externe renvoient des vecteurs d'événements qui sont ajoutés à la liste générale des événements. La seule tâche de programmation qui incombe au spécialiste du domaine, est de spécifier la dynamique des modèles atomiques à travers ses quatre méthodes. Une fois le modèle atomique créé, il est stocké dans la bibliothèque pour être utilisé plus tard dans une composition de modèle à l'intérieur d'un modèle couplé. Les compositions de modèles sont réalisées à travers l'interface graphique de modélisation présentée dans la section suivante.

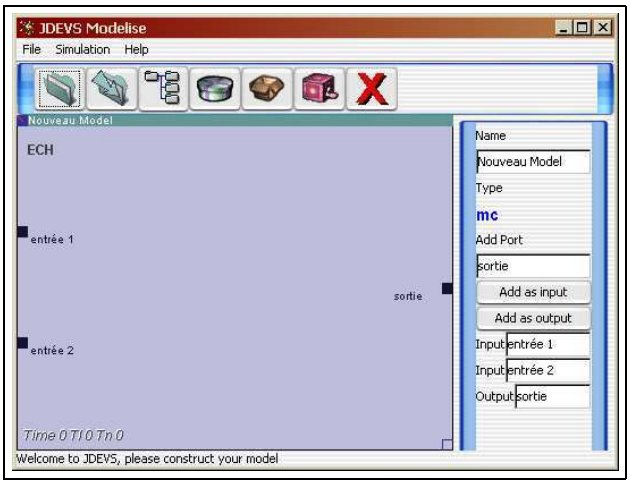

FIG. 5.3*: Interface de modélisation en diagramme contenant un modèle couplé et son panneau de propriétés*

5.1.2 Interface graphique de modélisation

L'interface de modélisation en diagrammes permet aux utilisateurs de construire graphiquement leurs modèles. La figure 5.3 représente une vue de l'outil constitué d'un modèle couplé avec deux ports d'entrée et un port de sortie. L'implémentation de cette interface est conforme au cadriciel, ainsi chaque modèle est présenté dans un espace de travail (partie centrale de la figure 5.3) et est associé à un panneau de propriétés (situé à droite dans la figure 5.3). De plus, l'interface dispose d'une barre de menu simple permettant d'ajouter des modèles vides (atomique, couplés ou d'autres techniques de modèlisation), d'ouvrir une bibliothèque et de supprimer des modèles. Modèles atomiques et couplés ont des panneaux de propriétés différents, la figure 5.3 présente celui d'un modèle couplé permettant d'ajouter et de supprimer des ports et de changer son nom.

Le panneau de modèles atomiques figure 5.4 permet d'inspecter et de modifier la valeur des paramètres du modèle, de lancer l'éditeur correspondant (ici éditeur de code Java) et de compiler le modèle afin de le recharger dans l'interface graphique.

Les composants peuvent être déplacés et redimensionnés à l'aide de la souris dans ce canevas. Un port est représenté par une poignée noire dans un composant : les couplages s'effectuent en cliquant dans les ports source puis destination.

Pour ajouter des modèles existant il suffit de les glisser depuis le composant bibliothèque 5.5

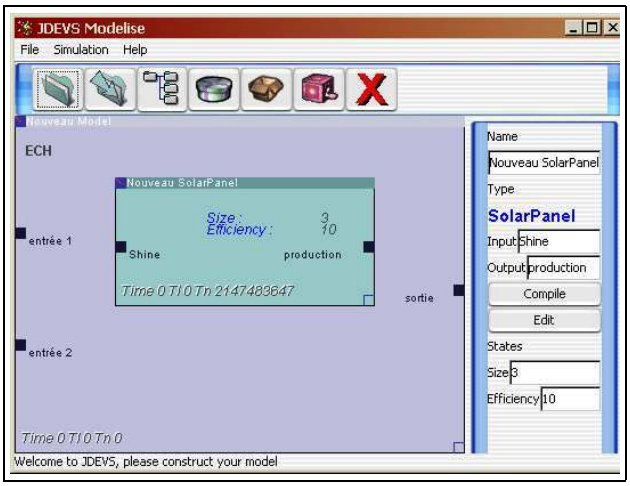

FIG. 5.4*: Interface de modélisation en diagramme contenant un modèle atomique et son panneau de propriétés*

vers l'espace de travail. La bibliothèque est l'élément essentiel permettant le stockage de modèles ; il est brièvement présenté dans la section suivante.

5.1.3 Stockage

La partie visible du module de stockage de JDEVS est un composant graphique présenté en figure 5.5 ; ce composant présente les modèles par domaine et sous-domaines sous la forme d'un arbre.

Le module de stockage permet de formater la structure des modèles pour pouvoir l'enregistrer au format XML et la stocker dans une bibliothèque du type HmLib. Une description détaillée de l'architecture de bibliothèque HmLib est disponible dans [Bernardi, 2002]. Le but d'une telle bibliothèque est d'offrir un moyen facile et performant de stocker et de récupérer les modèles. La bibliothèque HmLib peut aussi être considérée comme une base de données orientée objet, ce qui facilite aussi le stockage des modèles en XML. En plus de la structure, il est aussi possible de stocker les liens d'héritage et d'abstraction entre modèles. Les modèles stockés dans la bibliothèque (habituellement sous la forme de code source) sont appelés "hors-contexte". Pour récupérer un modèle, il faut l'instancier puis le mettre "en-contexte" en le replaçant dans l'état où avait été stocké l'objet. Plusieurs scénari de simulation peuvent ainsi créer plusieurs modèles "en-contexte" à partir

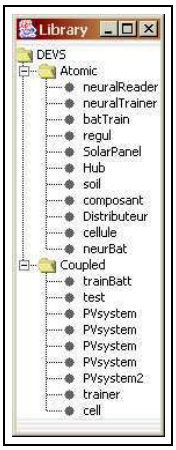

FIG. 5.5*: Composant de l'interface graphique présentant les modèles disponibles dans la bibliothèque*

d'un modèle "hors-contexte". C'est à partir de modules "en contexte" qu'est lancée la simulation dans les cadres expérimentaux présentés dans la section suivante.

5.1.4 Cadres expérimentaux

Les cadres expérimentaux sont les interfaces utilisateur de simulation des modèles stockés dans la bibliothèque. Les expérimentations sur des modèles en diagramme peuvent être effectuées directement depuis l'interface graphique de modélisation. Il est ainsi possible de vérifier le comportement d'un modèle en cours de création et donc d'ajuster ses paramètres ou corriger les erreurs de conception sans explicitement utiliser le module cadre experimental. L'application permettant d'exécuter les expérimentations se présente avec la même interface que l'interface de modélisation, chaque modèle possède un panneau de propriétés. Le panneau de propriété affiché est le panneau du modèle actif (i.e. le composant sélectionné).

La figure 5.6 présente un modèle en cours de simulation dans l'interface. Le lancement de la simulation depuis l'interface graphique permet de charger une liste d'événements d'entrée et de spécifier les paramètres de l'expérimentation ; ces actions sont effectuées depuis le panneau de simulation (figure 5.6, en haut à gauche). L'option "track simulation" permet de réaliser la simulation en insérant un temps d'attente entre l'envoi de deux événements d'entrée pour vérifier le comportement du modèle. La liste des événements à traiter est affichée à l'intérieur de chaque mo-

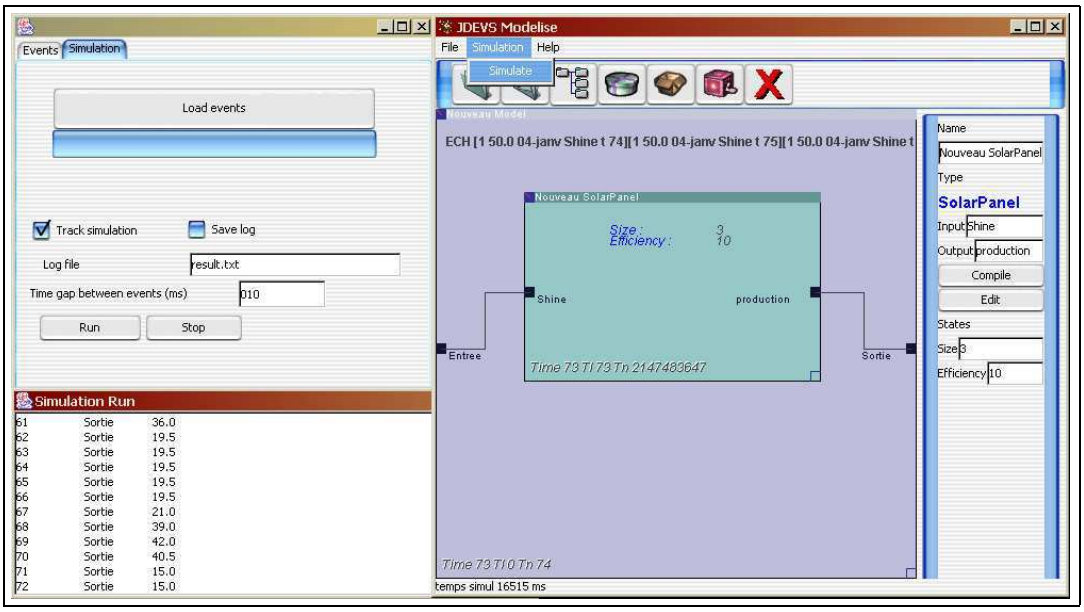

FIG. 5.6*: Interface de modélisation en diagramme de l'environnement JDEVS*

dèle couplé et évolue au cours de la simulation. Il est possible de changer les paramètres de chaque modèle atomique durant la simulation en l'activant puis en modifiant les valeurs à l'intérieur de son panneau de propriétés (figure 5.6, à droite). Le résultat de simulation est une liste d'événements de sortie qui peut être soit affichée (figure 5.6, en bas à gauche), soit sauvegardé dans un fichier.

Dans le cas de simulation de modèles spatialisés, il est peu commode de laisser à la charge de l'utilisateur de recomposer des cartes à partir d'événements pour pouvoir les analyser. Nous avons pour cela développé un module de connexion au SIG présenté dans la section suivante.

5.1.5 Interconnexion au SIG

[Brandmeyer et Karimi, 2000] ont détaillé plusieurs méthodologies de couplage au SIG parmi lesquelles :

- l'intégration totale, où l'environnement de simulation est développé comme un module du SIG et modifie directement les données pendant la simulation,
- le couplage "serré" où chaque changement d'attribut du modèle pendant la simulation déclenche une remise à jour de la base de donnée,

– le couplage "lâche" où les données d'initialisation sont importées depuis le modèle et les résultats exportés vers le SIG à la fin de la simulation.

L'intégration totale a été écartée car notre souci était de conserver la plus grande indépendance possible de notre logiciel vis à vis l'égard de logiciels tiers.

Le couplage serré, même s'il constitue la meilleure solution, s'est vite avéré impraticable car il génère de grands ralentissements dus à des accès trop fréquents à la base de donnée.

Nous avons donc opté pour un couplage de type "lâche". Pour cela JDEVS utilise la bibliothèque Geotools [Geotools, 2003] qui permet de stocker et de lire les formats de fichiers Arcview, ASCII and GML (Geographic Markup Language).

L'importation de données depuis le SIG ne pose pas de problème majeur car les seules cartes nécessaires sont les cartes d'initialisation qui représentent un instantané des attributs d'un domaine dans le temps. L'exportation des résultats de simulation est plus délicate car les SIG ne sont pas spécialement conçus pour stocker des attributs temporels. Dans le cas de JDEVS le résultat d'une simulation est une suite d'événements comme présenté en figure 5.1.

time	port	value
5	out[1-3]	1.0
10	out[2-4]	5.0
12	out[0-1]	1.0
16	out[1-4]	7.0
22	out[2-2]	5.0
25	out[3-1]	3.0

TAB. 5.1*: Liste d'événements de sortie d'une simulation*

Où "time" est le temps d'arrivée de l'événement, "port" est une référence sur le port et la cellule où s'est effectué le changement et "value" la nouvelle valeur. La carte n'est donc pas entièrement remise à jour à intervalle de temps réguliers. Pour permettre l'exportation de résultats il faut donc reconstituer des cartes raster à partir des événements de sortie. Il faut donc choisir l'intervalle de temps désiré entre deux cartes de sortie, le module de connexion au SIG les reconstituant en appliquant tous les changements depuis l'initialisation. Le tableau 5.2 présente les cartes correspondantes aux événements présentés en 5.1 avec un intervalle de temps de 5 unités.

T5	T10	T15	T20	T25
00000	00000	01000	01000	01000
00000	00000	00000	00000	00530
01000	01000	01000	01000	01000
00000	00500	00500	07500	07500
00000	00000	00000	00000	00000

TAB. 5.2*: Exemple de génération de cartes à partir de liste d'événements*

Cet ensemble de cartes raster peut ensuite être importé dans un SIG. Le module de connexion au SIG est, en pratique, un module intégré à tous cadres expérimentaux. Cependant, l'interface graphique de JDEVS ne constitue pas le seul cadre expérimental disponible pour réaliser la simulation de modèle. Comme défini dans le cadriciel, il est possible de générer des applets de simulation pour satisfaire les utilisateurs finaux n'ayant pas besoin de contrôle total sur la simulation. Les applets n'ont toutefois été développées que pour les logiciels résultant de l'intégration des techniques dédiées aux modèles spatiaux. Le module de simulation présenté en figure 5.6 est donc le seul cadre expérimental pour les techniques Classic-DEVS et Feedback-DEVS présentées dans la section suivante.

5.2 Modélisation avec Feedback-DEVS

La modélisation de modèles avec Feedback DEVS suit le même processus que la modélisation avec Classic-DEVS. La seule différence concerne le squelette de code généré qui permet l'ajout de ports de type "feedback" et la méthode "react". La figure 5.7 présente le squelette du code créé pour

$DevsFAtom = < X(i1), F(f1), S(A), Y(o1), \delta_{int}, \delta_{ext}, \delta_{react}, \lambda, t_a >$

Le protocole d'expérimentation de modèles Feedback-DEVS est identique à celui des modèles Classic-DEVS. Les résultats de simulation sont ici aussi une suite d'événements qui peuvent être enregistrés dans un fichier pour un post-traitement par un logiciel de statistique tiers. Aucun cadre de simulation spécifique n'a été développé pour les modèles Feedback-DEVS car le module de simulation permet déjà de présenter toutes les informations du modèle en cours de simulation de

```
public class DevsFAtom extends FeedbackModel {
      Port i1 = new Port(this,"i1","IN");
      Port f1 = new Port(this,"f1","FEEDBACK");
      Port o1 = new Port(this,"o1","OUT");

public DevsFAtom () { ]
      super("DevsFAtom");
      states.setProperty("A",""); }
    EventVector outFunction(Message m) {
         return new EventVector();}
    void intTransition() {}
    EventVector extTransition(Message m) {
         return new EventVector();}
    EventVector react(Message m) {
         return new EventVector();}
    int advanceTime(){return A;}
}
```

FIG. 5.7*: Code source JAVA pour un composant Feedback-DEVS*

manière satisfaisante. Ce n'est pas le cas pour tous les types de modèles, en particulier pour ceux ayant des attributs spatiaux. Ce type de modèle est pris en compte dans les extensions du logiciel pour les modèles spatialisés. La première de ces extensions est présentée dans la section suivante, la modélisation par automates cellulaires.

5.3 Modélisation par automates cellulaires

La modélisation par automates cellulaires permet de modéliser le comportement d'un domaine en le morcelant en zones de comportements identiques, les cellules. Pour permettre l'intégration de cette technique à JDEVS, nous avons défini une architecture standard de ce type de modèles présentée en figure 5.8.

Dans la plupart des modèles que nous avons eu à traiter, les données d'entrée devaient être identiques pour toutes les cellules, c'est pour cela que chaque cellule est connectée en entrée à un modèle *distributeur* chargé d'envoyer les messages d'entrée à toutes les cellules. Tous les ports de sortie des cellules sont connectés à un modèle *concentrateur*, chargé d'acheminer tous

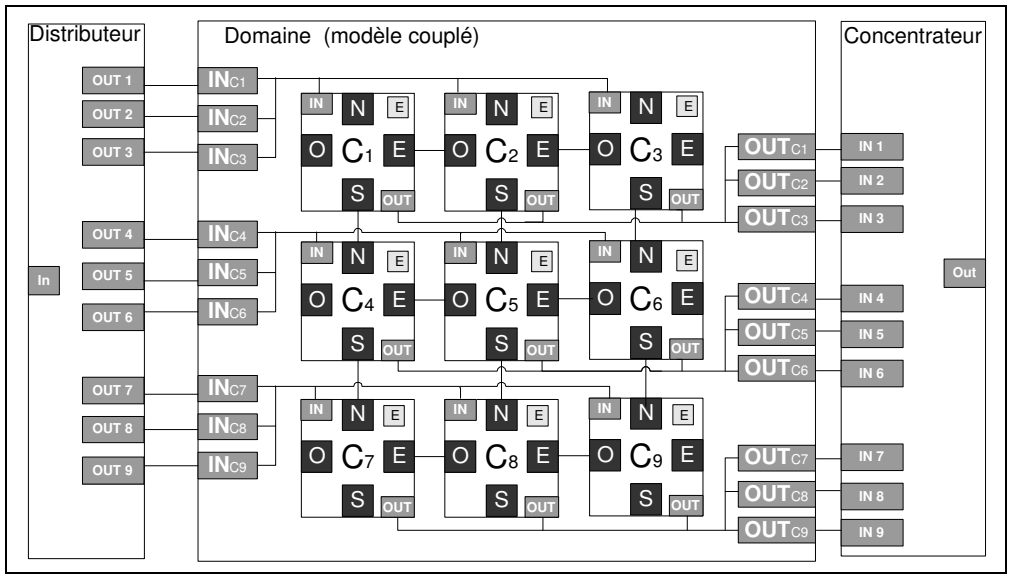

FIG. 5.8*: Architecture standard des modèles cellulaires dans JDEVS*

les événements reçus depuis les cellules vers un unique port de sortie. L'utilisation des modèles *distributeur* et *concentrateur* n'est toutefois pas obligatoire ; il suffit alors de connecter directement les ports d'entrée ou de sortie d'une cellule à un port externe du modèle. Cette section présente l'extension de JDEVS, composée des modules spécifiques à l'interface graphique, au stockage et aux cadres expérimentaux.

5.3.1 Interface graphique de modélisation par automates cellulaires

La création d'un modèle cellulaire implique la création d'une cellule comme composante du domaine. Un modèle de cellule est un modèle atomique standard composé de ports de connexion aux cellules voisines (Au minimum nord, sud, est et ouest). Pour créer un modèle cellulaire dans JDEVS il faut ajouter un modèle cellulaire vide, cette action génère un squelette de code java pour la cellule générique du domaine. Ce squelette contient au minimum cinq ports d'entrée et autant de sortie :

$< X\{N,S,E,O,in\}, Y\{N,S,E,O,out\} >$

Ces ports correspondent aux ports de connexion au voisinage et aux ports d'entrée et de sortie généraux. Le squelette de code définit aussi les fonctions vides correspondant aux méthodes DEVS

standard permettant de définir le comportement général. Une fois le comportement de la cellule décrit en code Java, le modèle doit être mis en contexte pour permettre le couplage avec une cellule en particulier. Pour mettre en contexte un modèle cellulaire il faut charger les cartes nécessaires exportées d'un SIG au moment de son initialisation. Chaque instance de cellule créée prend alors le suffixe [numéro ligne-numéro colonne] comme nom d'identification. Par exemple, une instance d'un modèle atomique cellule nommé "cell" situé à la ligne 12 et à la colonne 8 prendra le nom "cell[12-8]".

Le modèle mis en contexte se représente sous la forme d'un modèle avec les ports d'entrée et de sortie généraux des *distributeur* et *concentrateur* comme nous l'illustrons dans la figure 5.8. Le panneau de contrôle des modèles cellulaires permet de charger les cartes d'initialisation des paramètres du modèle, chaque carte correspond à un attribut des modèles atomiques cellules. Une carte est un lien URI vers un fichier raster ASCII, ce qui permet d'accéder à des ressources partagées sur un serveur d'informations géographiques.

La visualisation de l'état d'un modèle en cours de simulation est importante pour verifier son comportement. Pour cela, il est possible de spécifier les attributs du modèle devant être affichés ainsi que la palette de couleurs associée à ces attributs. Une palette est composée en associant une couleur à un état dans lequel peut se trouver le modèle.

Le couplage entre différents modèles se fait à l'intérieur de l'interface de modélisation. Il est possible de lier un port externe à une cellule directement en la sélectionnant et en choisissant le port approprié. Les structures ainsi créées sont enregistrées dans un fichier XML par le moteur de stockage décrit dans la section suivante.

5.3.2 Stockage de modèles cellulaires

Le stockage de la structure des modèles cellulaires s'effectue dans un fichier XML pouvant être stocké dans une bibliothèque. Le format du fichier XML résultant, est présenté en figure 5.9 ; il est compatible avec le format standard de stockage de modèles couplés et ajoute la balise `TERRAINDATA` contenant un lien `URI` vers les cartes destinées à la mise en contexte du modèle. La

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE MODEL [
      <!ELEMENT MODEL (TYPE, NAME, CLASS?, BOUNDS?, TERRAINDATA*, INPUT*,
            OUTPUT*, CHILD*, EIC?, EOC?, IC?, EXECUTIVE?)>
           <!ELEMENT NAME (#PCDATA)>
            <!ELEMENT TYPE (#PCDATA)>
            <!ELEMENT CLASS (URI)>
            <!ELEMENT BOUNDS (LOCX, LOCY, WIDTH, HEIGHT)>
            <!ELEMENT LOCX(#PCDATA)>
            <!ELEMENT LOCY (#PCDATA)>
            <!ELEMENT TERRAINDATA (URI)>
            <!ELEMENT WIDTH (#PCDATA)>
            <!ELEMENT HEIGHT (#PCDATA)>
            <!ELEMENT PORT (URI?, #PCDATA)>
            <!ELEMENT URI (#PCDATA)>
            <!ELEMENT CHILD (MODEL* | URI*)>
            <!ELEMENT INPUT (PORT*)>
            <!ELEMENT OUTPUT (PORT*)>
            <!ELEMENT LINK (PORT, PORT)>
            <!ELEMENT EIC (LINK*)
            <!ELEMENT EOC (LINK*)>
            <!ELEMENT IC (LINK*)>
            <!ELEMENT EXECUTIVE (MODEL* | URI*)> ]
>
```

FIG. 5.9*: Document de définition de format XML des modèles cellulaires*

balise `CLASS` représente ici le lien URI vers la classe du modèle atomique de la cellule composant le domaine.

En somme, les domaines créés et stockés dans la bibliothèque au format XML peuvent être directement simulés dans le module de simulation. Cependant cette interface n'est pas destinée à un utilisateur qui désire effectuer des expérimentations plus simples et sans contrôle sur un modèle. Pour cela, plusieurs cadres expérimentaux ont été développés pour les modèles cellulaires, ils sont présentés dans la section suivante.

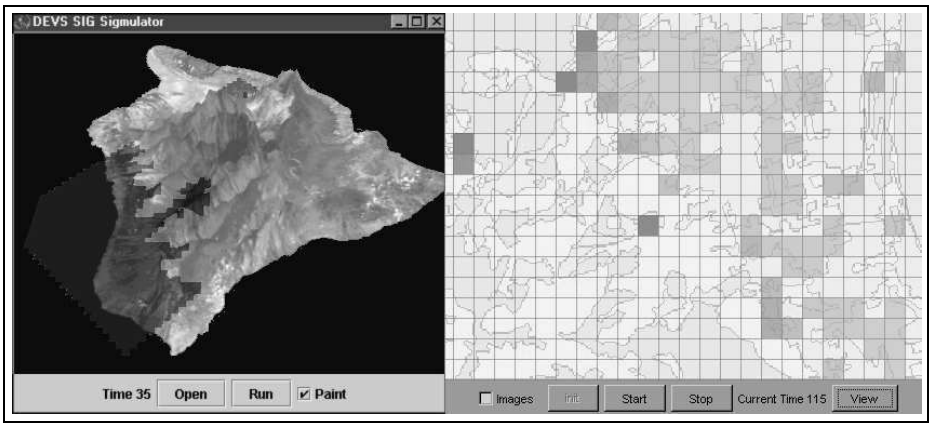

FIG. 5.10*: Cadres expérimentaux 3d et 2d de simulation de modèles cellulaires*

5.3.3 Cadres expérimentaux pour les modèles cellulaires

La simulation de modèles cellulaires peut s'effectuer directement à partir de l'interface de JDEVS. Cette interface peut toutefois se révéler trop lourde, c'est pourquoi nous avons réalisé deux autres cadres d'expérimentation pour les modèles cellulaires : l'un permettant la visualisation en deux dimensions, l'autre en trois dimensions.

Les panneaux de modélisation cellulaire permettent non seulement d'effectuer la simulation de modèles cellulaires de manière simplifiée mais de plus, cette simulation est interactive : il est en effet possible d'envoyer des événements prédéterminés en direction d'une seule cellule, en cliquant simplement sur la carte.

Les panneaux de modélisation sont présentés en figure 5.10. Les modèles sont initialisés en utilisant le connecteur SIG décrit dans la section suivante. Les cellules sont instanciées et liées directement par le parser XML du document de définition du modèle. Ce modèle est ensuite passé au simulateur qui lance la simulation. La visualisation des phénomènes en cours de simulation est effectuée soit sur une carte en deux dimensions avec le panneau de simulation 2d (figure 5.10 droite), soit sur une carte en trois dimensions effectuée dans le panneau de simulation 3d (figure 5.10 gauche). Dans le cas d'une visualisation 3d, il est nécessaire de spécifier une carte raster d'élévation pour le domaine à simuler.

Ces panneaux de simulation peuvent aussi être exportés sous la forme d'applets et publiés sur le web. La publication sur le web offre une gamme d'utilisation plus large à JDEVS, car elle permet

à des utilisateurs non initiés d'utiliser des modèles scientifiques rigoureux de manière simple. Pour cela le modélisateur définit un domaine d'étude et le modèle à publier puis diffuse l'applet ainsi que le fichier XML du modèle associé. Le but final des simulation de systèmes naturels est souvent la prise de décision dans le cadre de l'aménagement du territoire. Les personnes habilitées à prendre des décisions sont souvent des responsables politiques : leur permettre de s'approprier le modèle en effectuant eux-mêmes les expérimentations permet d'obtenir un plus fort impact.

Pour des simulations dont le résultat est destiné à un post-traitement, les résultats sont exportés vers un SIG par le module de connexion.

La modélisation par automates cellulaires n'est pas l'unique moyen disponible dans JDEVS pour représenter des phénomènes spatialisés. La section suivante présente l'intégration de l'autre technique, Vector-DEVS, telle qu'elle à été décrite dans le chapitre précédent.

5.4 Modélisation avec Vector-DEVS

L'utilisation de modèles Vector-DEVS n'a pas encore été testée dans le cadre de la modélisation d'un système physique. Ce manque d'expérience fait que nous n'avons pas pu définir les interfaces utilisateurs sans la contribution indispensable des utilisateurs finaux.

Toutefois, l'implémentation de Vector-DEVS dans JDEVS a été réalisée en suivant l'architecture de classe définie dans le cadriciel. En conséquence, le module est encore en développement et nous ne disposons que de la base nécessaire à des expérimentations limitées.

La figure 5.11 présente le cadre expérimental actuel de Vector-DEVS, il nous permet d'afficher les modèles en cours d'exécution et d'effecteur des expérimentations sur des cartes de type vecteur, importées grâce à GeoTools [Geotools, 2003].

La connexion à d'autres modèles se fait actuellement par la définition de couplages directement en code JAVA. Ainsi, nous étudions la possibilité d'appliquer cette technique à la modélisation de propagation de feu de forêts. Cette expérience permettra de continuer le développement d'un module complet JDEVS pour la modélisation visuelle et interactive de modèles Vector-DEVS.

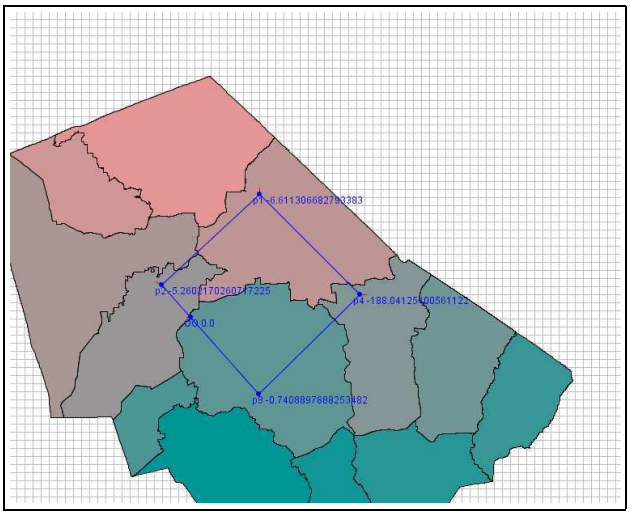

FIG. 5.11*: Cadre expérimental de simulation Vector-DEVS*

5.5 Conclusion

Cette section a présenté JDEVS, implémentation logicielle du cadriciel générique de modélisation et de simulation. A travers ce développement nous avons pu démontré la validité de notre cadriciel et l'intêret de son utilisation comme base de construction, d'environnements de modélisation. Plusieurs techniques ayant été ajoutées au logiciel, ceci nous autorise à l'exploiter comme plate-forme de développement de modèles, dans le domaine de la modélisation et de la simulation de systèmes naturels complexes.

Des expérimentations diverses furent réalisés avec JDEVS tout au long de l'étude à l'aide de JDEVS ; elles sont présentées dans le chapitre suivant : l'expérimentation.

CHAPITRE 6

Expérimentation

CE CHAPITRE présente un aperçu des modèles développés à l'aide de JDEVS. Notre but n'est pas ici de montrer la validité de ces modèles mais plutôt d'illustrer la démarche de multi-modélisation pouvant être suivie en utilisant cet outil. Nous ne présenterons donc pas de résultats détaillés et comparés sur ces modèles. La plupart sont d'ailleurs en cours de validation ou d'évaluation. Les trois modèles sont proposés dans trois sections distinctes de ce chapitre. Le premier, un modèle de centrale photovoltaïque propose l'utilisation de Feedback-DEVS pour l'implémentation d'un réseau de neurone comme sous modèle de batterie. Le second, un modèle de dispersion d'insecte illustre le processus de modélisation par automates cellulaires. Enfin, le dernier modèle, de propagation de polluants, est composé de sous modèles à automates cellulaires pour la dimensions spatiale du système, et de Feedback-DEVS pour un sous-modèle empirique de calcul de concentration en nitrates.

Le lecteur intéressé trouvera en annexe les logiciels dérivés présentés dans ce chapitre.

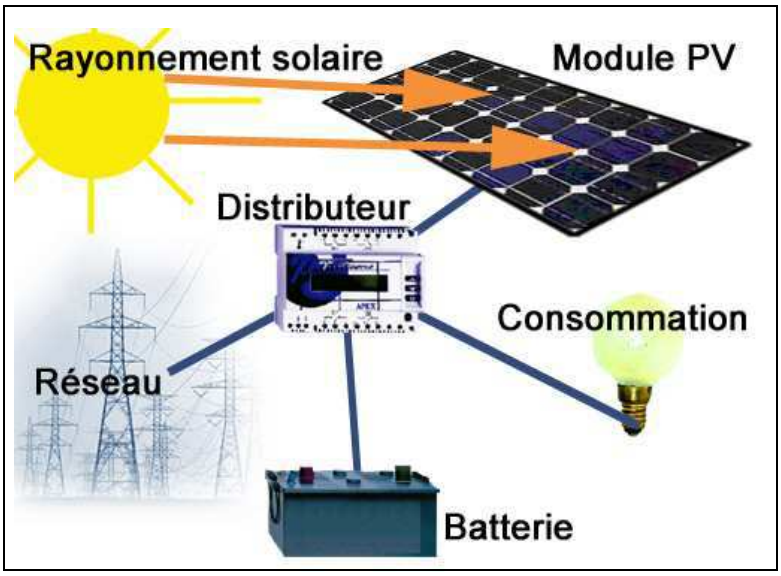

FIG. 6.1*: Un système photovoltaïque*

6.1 Modélisation d'un système photovoltaïque

Les systèmes à énergie solaire photovoltaïque, présentés en figure 6.1, sont aussi connus comme systèmes "PV". Ce sont des centrales électriques en miniature pouvant fonctionner en mode autonome ou mode connecté au réseau électrique.

Les systèmes "PV" autonomes sont composés d'une batterie optionnelle de panneaux de production d'électricité photovoltaïque et d'un répartiteur [Jungst et al., 2000]. Leur mode de fonctionnement est simple : le rayonnement solaire est converti en électricité par le panneau photovoltaïque, l'énergie produite est ensuite partagée par le distributeur entre la recharge de la batterie et les postes de consommation. Si le soleil ne brille pas suffisamment, la batterie est utilisée pour fournir la puissance manquante. Enfin si la batterie est complètement chargée, l'énergie non consommée est perdue.

Les systèmes "PV" connectés utilisent le réseau électrique comme batterie, ils sont uniquement composés d'un répartiteur et de panneaux photovoltaïques. Dans un système connecté, l'énergie produite peut être renvoyée sur le réseau tandis que pendant les périodes de sous production, le réseau complète l'apport énergétique, assurant du même coup un fonctionnement sans interruption de service.

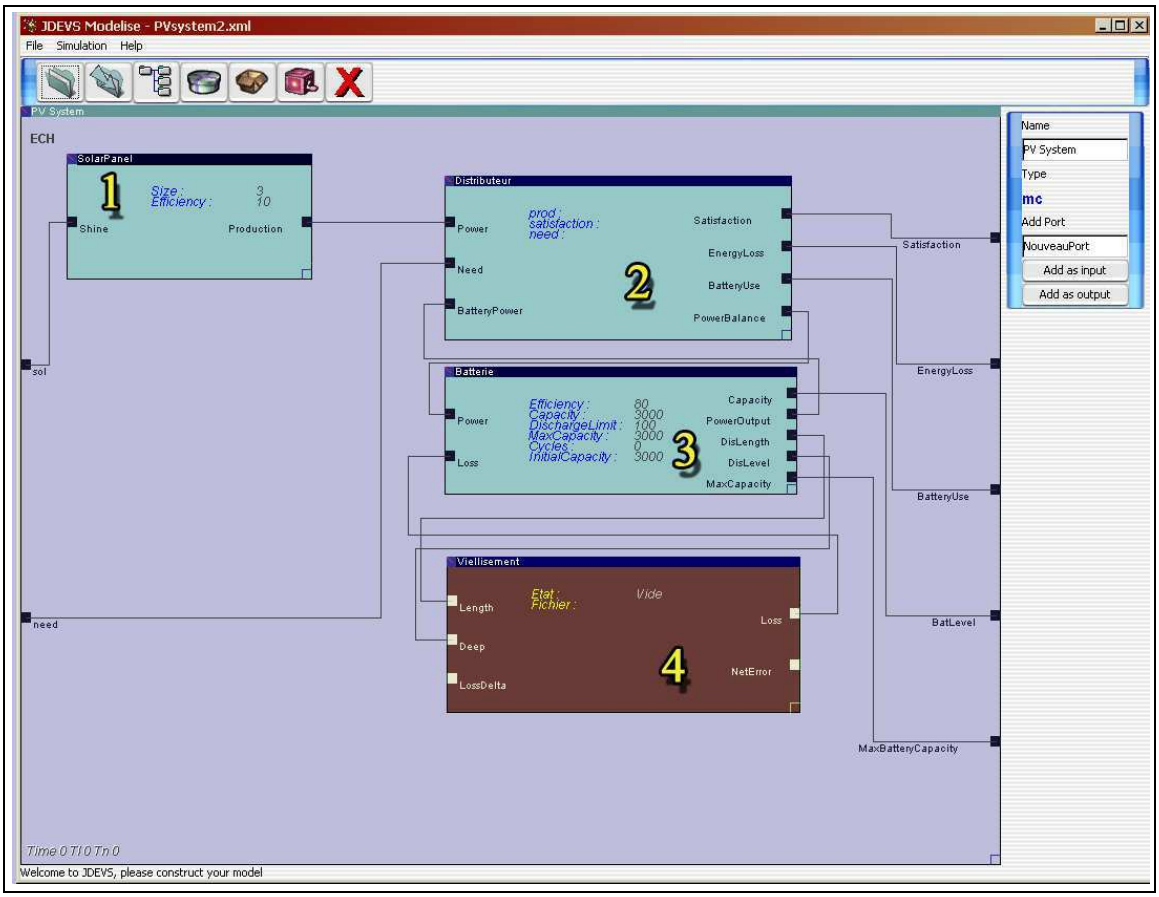

FIG. 6.2*: Modèle du système PV dans JDEVS*

Le but final de ce modèle est d'effectuer des expérimentations virtuelles sur ces systèmes et aider à leur dimensionnement en fonction des sites d'implantation. Il a été développé en collaboration avec l'équipe énergétique de notre laboratoire qui traite cette problématique [Notton et al., 2000]. Cette équipe, en tant que spécialiste du domaine, a défini le protocole permettant de préciser le cadre expérimental et donc les paramètres et variables d'entrée et de sortie du modèle. La figure 6.2 présente le modèle implémenté dans JDEVS.

Le modèle général a deux entrées :

- `sol` : l'ensoleillement en Watt/Heure/mètre carré.
- `need` : le besoin en consommation en Watt/heure

Il est possible de suivre plusieurs variables de sorties du modèle suivant le cadre expérimental dans lequel l'utilisateur se place. Cependant la majorité des études dans le domaine énergétique (par exemple [Halupka et al., 2000] ou [Notton et al., 2000]) s'intéresse à un nombre restreint de ces variables. Nous avons défini cinq ports de sortie pour permettre de suivre l'évolution du modèle :

- `Satisfaction` : le pourcentage de besoin satisfait par le système par rapport au besoin réel de consommation.
- `EnergyLoss` : la quantité d'énergie solaire perdue en Watt/heure.
- `BatteryUse` : le pourcentage d'utilisation d'énergie provenant de la batterie par rapport à l'énergie du panneau photovoltaïque.
- `BatLevel` : le niveau de la batterie en Watt (pour une tension constante de 12 volts).
- `MaxBatteryCapa` : la capacité maximum de la batterie en Watt (pour une tension constante de 12 volts).

Une des contraintes posées par les spécialistes du domaine et les utilisateurs finaux est la modalité de mesure des données d'entrée et de sortie. En effet, la grande majorité des stations de mesures, comme les modèles existants, fonctionnent en temps discret.

Le modèle doit donc être capable de recevoir en entrée des événements arrivant à intervalles de temps réguliers (ici l'heure). Les événements de sortie sur les ports `EnergyLoss` et `BatLevel` doivent aussi être générés à intervalles de temps régulier.

Les autres ports de sortie sont plus conformes à un mode de simulation à événements discrets et ne sont actifs que pour informer d'un changement de valeur. Les sous-modèles peuvent ainsi fonctionner sur ce mode mixte temps/événements discrets. Ils sont présentés en détail dans la suite de cette section.

6.1.1 Le panneau photovoltaïque

Le modèle du panneau photovoltaïque (Figure 6.2(**1**)) représente le système de production d'énergie, il possède deux ports :

– Un port d'entrée, `Shine` : l'ensoleillement en Watt/Heure/mètre carré.

– Un port de sortie, `Production` : la production électrique en watt/heure.

Nous avons défini deux paramètres dans ce modèle pour permettre à l'utilisateur final d'effectuer plusieurs expérimentations :

– `Size` : la taille en mètres carrés,

– `Efficiency` : le rendement en % de conversion énergétique (solaire vers électrique).

Le comportement des cellules photovoltaïques composant un panneau est bien connu, il peut donc être modélisé comme un modèle atomique DEVS classique, en utilisant les données du fabricant. L'énergie de sortie calculée par le modèle est envoyée au distributeur.

6.1.2 Le distributeur

Le modèle du distributeur (Figure 6.2(**2**)) est aussi un modèle DEVS classique ; ce modèle est chargé d'assurer la répartition des charges entre la batterie, le panneau solaire et les postes de consommation. Ce sous-modèle est dans une position centrale car il est connecté en entrée, à la batterie, au panneau photovoltaïque et aux postes de consommation par les ports :

– `Power` : la production en Watt/heure du système source d'énergie,

– `Need` : la demande électrique en watt/heure.

– `BatteryPower` : l'énergie disponible dans la batterie, en Watt.

En sortie ce modèle est connecté à la batterie pour communiquer le besoin en énergie. Les autres ports de sortie sont utiles à la génération de données pour l'analyse des résultats de simulation. Le sous-modèle est globalement composé de quatre ports de sortie :

- `Satisfaction` : le pourcentage de besoin satisfait par le système par rapport au besoin réel de consommation,
- `EnergyLoss` : la quantité d'énergie solaire perdue en Watt/heure,
- `BatteryUse` : le pourcentage d'utilisation d'énergie provenant de la batterie par rapport à l'énergie du panneau photovoltaïque,
- `PowerBalance` : le besoin en énergie (décharge) ou bien la quantité d'énergie disponible pour la recharge de la batterie (si la batterie n'est pas pleine), en Watt.

Dés que le niveau change dans le modèle de la batterie, un message est envoyé au distributeur pour l'informer de la nouvelle valeur ; ainsi le distributeur connaît constamment le niveau d'énergie dans la batterie. Lorsqu'il y a un besoin en énergie et que l'énergie disponible est suffisante, un message est envoyé par le port `PowerBalance`, donnant la quantité d'énergie à enlever à la batterie. Lorsqu'il y a un surplus d'énergie et que le niveau de la batterie n'est pas au maximum, un message est envoyé pour la recharge de la batterie.

Le comportement de la batterie évolue durant la simulation à cause de son vieillissement. Ce vieillissement est souvent propre à un type ou même une marque de batterie. Il est alors difficile de le modéliser avec des techniques conventionnelles. Le modèle de batterie qui est présenté dans la section suivante utilise un modèle Feedback-DEVS comme modèle de vieillissement.

6.1.3 La batterie et le modèle de vieillissement

Le comportement physique d'une batterie est très complexe et fait intervenir de nombreuses réactions chimiques qui nécessitent beaucoup de données pour pouvoir être simulées de manière conventionnelle. Nous avons donc adapté un modèle existant, décrit dans [Jungst et al., 2000], afin d'obtenir des résultats satisfaisants. Dans ce modèle, la batterie est considérée comme un réservoir d'électricité avec une capacité et une efficacité de charge en paramètres et peut donc être modélisée comme un modèle DEVS classique. A chaque batterie est attaché un modèle de vieillissement qui

modifie sa capacité de charge en cours de simulation. Le modèle de vieillissement est ici un modèle Feedback-DEVS qui implémente un réseau de neurones [Filippi et al., 2001] utilisant la technique de rétro-propagation du gradient [Hecht-Nielsen, 1989],[Carling, 1992].

Selon [Haykin, 1994] "*Un réseau de neurones est un processeur massivement distribué en parallèle qui a une propension naturelle pour stocker de la connaissance empirique (experiential knowledge selon l'auteur) et la rendre disponible à l'usage. Il ressemble au cerveau sur deux aspects : La connaissance est acquise par le réseau au travers d'un processus d'apprentissage. Les connexions entre les neurones, connues sous le nom de poids synaptiques servent à stocker la connaissance*."

Les réseaux de neurones informatiques se basent sur ces principes pour simuler le comportement d'un système lorsque les seules connaissances exploitables sont des données expérimentales. Toutefois un des principaux inconvénients des réseaux de neurones est qu'ils sont difficilement vérifiables [Stimpfl-Abele, 1995],[Dukelow, 1994]. Le lecteur intéressé trouvera dans [Hecht-Nielsen, 1989] ou [Filippi, 2000] une description détaillée de cette technique.

Ce modèle de vieillissement permet de calculer la perte de capacité à partir de la profondeur et de la durée de la décharge maximale. La batterie est considérée comme étant en décharge lorsque la puissance disponible est inférieure de moitié à la capacité maximale. Le modèle de vieillissement est présenté après une description du modèle de batterie.

Le modèle de batterie (Figure 6.2(**3**)) est ici considéré comme un réservoir de puissance mesurée en Watt. Il possède deux ports d'entrée :

- `Power` : la puissance qui doit être ajoutée ou retranchée à la quantité présente dans la batterie,
- `Loss` : la perte de capacité de stockage due au vieillissement.

De nombreuses variables de sortie sont nécessaires au calcul de vieillissement et à l'analyse du comportement de la batterie ; le modèle est composé de cinq ports de sortie :

- `Capacity` : la puissance disponible dans la batterie, en Watt, remise à jour dès que la capacité change. Ce port est destiné à être connecté à une sortie du modèle général pour permettre des analyses de résultats de simulation,

- `PowerOutput` : Identique au port précédent mais destiné au couplage avec les autres sous-modèles,
- `DisLength` : durée de la décharge en cours, exprimée en heures. Ce port émet toutes les heures dès que la puissance disponible est inférieure à la moitié de la capacité maximum,
- `DisLevel` : profondeur de la décharge en cours exprimé en pourcentage de la capacité initiale,
- `MaxCapacity` : capacité maximum de la batterie.

Le modèle de batterie dispose aussi de six paramètres dépendant des expérimentations et qui peuvent être ajustés avant ou au cours de la simulation :

- `Efficiency` : l'efficacité de charge et de décharge, exprimée en pourcentage.
- `Capacity` : la puissance courante disponible, exprimée en Watts.
- `DischargeLimit` : la limite de décharge ou seuil au delà duquel la batterie ne peut plus de se décharger, exprimée en Watt.
- `MaxCapacity` : capacité maximum de la batterie.
- `Cycles` : nombre de cycles charge/décharge.
- `InitialCapacity` : capacité initiale maximum de la batterie.

Le modèle utilisé pour calculer le vieillissement de la batterie est une implémentation en feedback-DEVS du modèle décrit dans [Jungst et al., 2000]. Il utilise la technique des réseaux de neurones à rétro-propagation du gradient pour estimer la perte de capacité d'une batterie en fonction de la profondeur et de la durée d'une décharge.

Un des avantages majeurs de l'utilisation du modèle décrit dans [Jungst et al., 2000] est que l'on dispose des données pour entraîner et valider le réseau de neurones. Ces données sont collectées au *Florida Solar Energy Center* [FSERC, 2003] qui dispose d'un grand centre d'étude sur les batteries. Cependant le modèle pose un problème lors de son implémentation dans un environnement de simulation à événements discrets car il est dirigé par une horloge. En effet, à cause des données disponibles nous ne pouvons calculer la perte de capacité que toutes les heures, en fonction de la durée de la décharge en cours. Le modèle de vieillissement n'est toutefois activé que pendant les périodes de décharges.

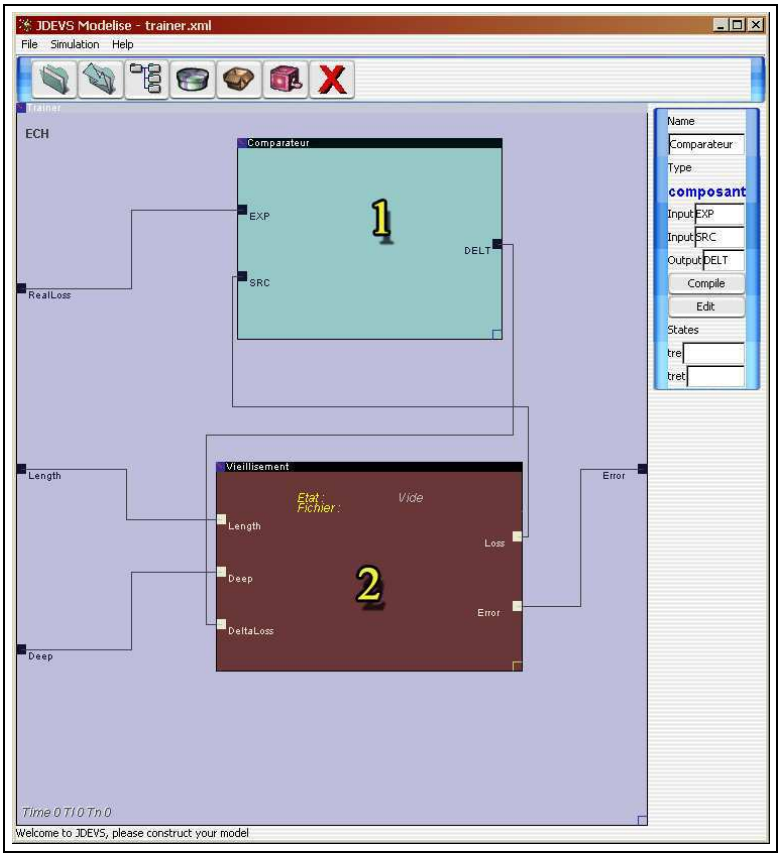

FIG. 6.3*: Modèle d'apprentissage du réseau de neurones chargé du calcul du vieillissement*

Le modèle utilise la technique Feedback-DEVS pour implémenter le réseau de neurones. Il possède deux ports d'entrée standards (Figure 6.2(4)) :

– `Length` : la durée de la décharge en cours,

– `Deep` : la profondeur de la décharge (en pourcentage).

Et deux ports de sortie :

– `Loss` : la perte de capacité de stockage due au vieillissement, exprimée en Watts,

– `NetError` : l'erreur de calcul du réseau de neurones.

Le modèle possède aussi un port d'entrée de Feedback, `LossDelta`, permettant d'effectuer l'apprentissage.

En figure 6.2(**3**) le port `LossDelta` n'est pas connecté car le modèle est déjà entraîné. Il est possible de sauvegarder les valeurs des poids du réseau de neurones directement dans un fichier pouvant être stocké dans la bibliothèque. Le modèle d'apprentissage est présenté en figure 6.3, où le modèle de *vieillissement* est noté 6.3(**2**). L'apprentissage du réseau de neurones est fait à l'aide d'un modèle *comparateur* (Figure 6.3(**1**)). Pendant l'apprentissage le modèle de vieillissement reçoit une valeur de durée et de profondeur de décharge, en même temps que le *comparateur* reçoit la valeur de la perte de capacité associée ; ces valeurs sont mesurées en laboratoire. Le modèle de vieillissement va alors programmer immédiatement l'envoi de la valeur de perte de capacité estimée au *comparateur*. La différence entre les deux valeurs constitue l'erreur qui est renvoyée au modèle de vieillissement par le port `LossDelta`, ce qui active la fonction de feedback effectuant l'apprentissage.

L'interface graphique de JDEVS permet de créer simplement et graphiquement la structure de tous ces modèles mais le comportement doit être spécifié en code Java. Le cadre expérimental de la simulation est ici identique à l'interface de modélisation. Les expérimentations, réalisées à partir d'une liste d'événements d'entrée et les différentes configurations, sont obtenues en modifiant les paramètres des modèles grâce à leurs panneaux de propriétés. Quelques résultats de simulations sont présentés dans la section suivante.

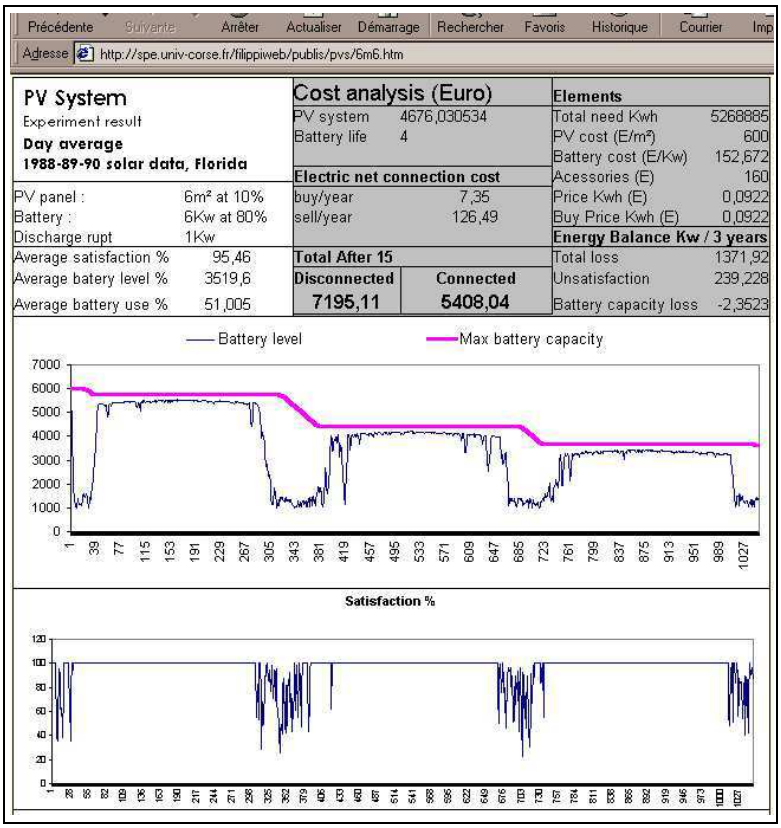

FIG. 6.4*: Page de présentation d'un résultat d'expérience*

6.1.4 Résultats de simulation

Dans le cadre d'une simulation standard de ce modèle, les données d'entrée sont fournies par l'utilisateur et les événements de sorties sont directement stockés dans des fichiers. Toutefois, grâce à la méthodologie utilisée, il est très simple d'intégrer ce modèle à l'intérieur d'un autre plus complexe comprenant, par exemple, un modèle de consommation et un modèle de calcul de pollution en fonction de l'énergie économisée.

Douze expérimentations ont été effectuées portant sur une période de trois ans et une combinaison de quatre tailles de panneaux différents et de trois capacités de batterie. Les profils de consommation ont été obtenus auprès du FSRC [FSERC, 2003], et correspondent à la consommation d'une maison avec cinq ampoules de type basse consommation utilisées 8h/jour, une télévision utilisée 5h/jour et un petit réfrigérateur (consommation totale journalière sous 12 Volts :

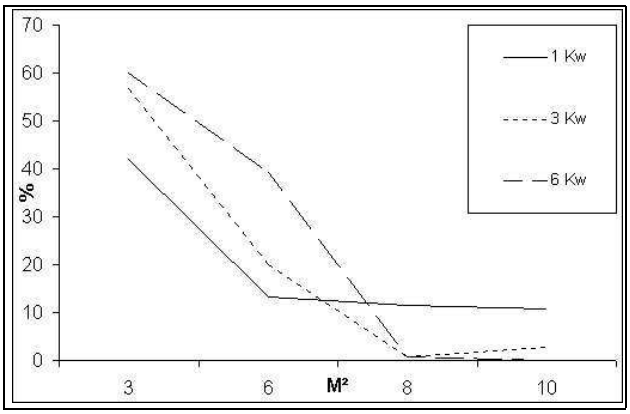

FIG. 6.5*: Perte de capacité de batterie après trois ans, en fonction de la taille du panneau photovoltaïque*

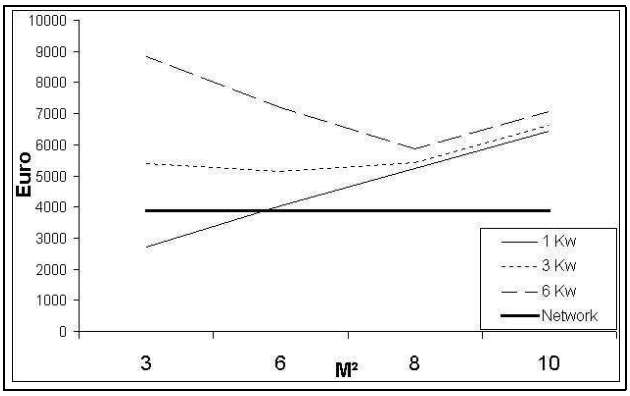

FIG. 6.6*: Prix d'un système PV déconnecté après 15 ans*

1600 Watts). Les données d'ensoleillement et des résultats plus détaillés de l'expérimentation sont disponibles dans [Filippi et al., 2001] et sur le CD-ROM en annexe 6.4.

La figure 6.4 présente une expérimentation d'une simulation avec une taille de panneau solaire de 6 mètres carrés et une puissance de 6 KW de batterie. L'effet du modèle de vieillissement est ici clairement visible ; la capacité maximum de la batterie est fortement dégradée chaque hiver.

La figure 6.5 montre que la perte de capacité de batterie après trois ans est dépendante de la taille du panneau photovoltaïque. Le calcul de coût d'un système photovoltaïque déconnecté est fortement dépendant de l'usure de la batterie car on considère qu'une batterie qui a perdu 40% de sa capacité doit être changée.

La figure 6.6 montre le prix cumulé d'un système connecté au réseau après 15 ans. Le coût

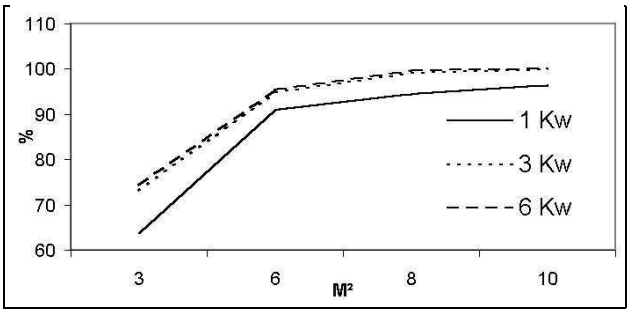

FIG. 6.7*: Satisfaction moyenne sur 3 ans*

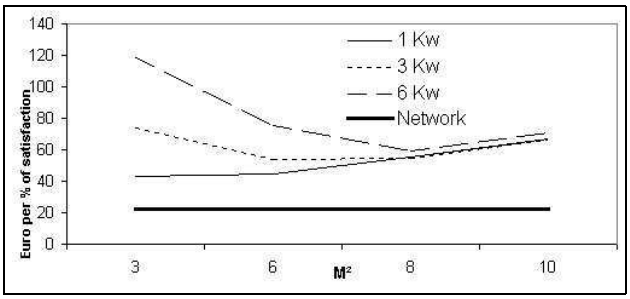

FIG. 6.8*: Prix par % de satisfaction pour un système déconnecté*

d'un système déconnecté est calculé en ajoutant le coût des batteries de rechange au coût initial de l'installation. Le réseau électrique peut être concurrencé si le système PV est bien dimensionné, mais cela se fait au détriment de la qualité de service (faible satisfaction).

La figure 6.7 présente la satisfaction moyenne. La satisfaction est un ratio entre la puissance demandée et la puissance fournie qui atteint 100% lorsque l'utilisateur ne manque jamais d'électricité. Un système PV doit être bien dimensionné pour offrir une satisfaction suffisante à de moindres coûts.

La figure 6.8 présente le coût d'une installation par % de satisfaction. Le réseau électrique est ici bien meilleur que les systèmes PV.

Bien que ce modèle produise des résultats exploitables, plus de données sont nécessaires pour entraîner le réseau de neurones de la batterie de manière plus satisfaisante. Il serait aussi nécessaire de simuler de manière plus adéquate les cellules photovoltaïques en prenant en compte la chaleur qui est un paramètre important dans le rendement et qui n'est pas pris en compte dans ce modèle. Le principal avantage de ce modèle réside dans sa flexibilité. Il est très facile, par exemple, de

FIG. 6.9*: Mouche des fruits méditerranéenne.*

remplacer graphiquement le composant "panneaux solaires" par un composant "éoliennes" et de créer ainsi un modèle de ferme éolienne. La section suivante présente l'utilisation de JDEVS pour un modèle utilisant une technique différente : la modélisation par automate cellulaire.

6.2 Modélisation de dispersion de mouches des fruits

Cette section présente un modèle de dispersion de mouches des fruits méditerranéennes. Ce modèle utilise la technique des automates cellulaires pour simuler la dispersion de mouches des fruits dans un espace à deux dimensions. La première partie de cette section présente brièvement le système de dispersion de la mouche des fruits avant d'en préciser le détail.

6.2.1 La mouche des fruits méditerranéenne

La mouche des fruits méditerranéenne (Ceratitis capitata) (figure 6.9) est l'un des fléaux les plus importants pouvant infecter les vergers. Si sa prolifération n'est pas contrôlée, la mouche des fruits peut infester la totalité d'une exploitation fruitière et notamment les fruits sensibles à son attaque tels que l'abricot et la pêche ; elle peut aussi gravement endommager les exploitations de pommes et d'agrumes. Parce qu'elle est non-polluante, la technique du SIT (Sterile Insect Technique) est utilisée de façon croissante pour éradiquer ce fléau. Cette technique consiste à relâcher un grand nombre de mâles stériles pendant un laps de temps suffisamment long dans le but de perturber fortement la reproduction de ces insectes.

L'un des objectifs principaux de ce modèle, développé à la demande du Service Régional de

Protection des Végétaux, est d'estimer la distribution géographique de mouches des fruits adultes, afin d'en prévoir les zones de concentration les plus fortes. Le comportement du modèle nous à été précisé par l'entomologiste désigné par le service de protection des végétaux de Haute-Corse pour lequel à été réalisée l'étude. L'intégralité du travail est disponible dans [Faure, 2001].

Les mouches adultes récemment émergées se nourrissent de substances sucrées présentes sur les arbres fruitiers. Les femelles ont en outre besoin de protéines afin de réaliser leur maturation sexuelle, dont la durée est assez courte (4 à 10 jours). Les mâles se rassemblent en groupes sur les plantes d'où ils émettent ensemble une phéromone sexuelle attirant les femelles. Peu après l'accouplement débute la ponte qui est fortement influencée par l'intensité lumineuse et a lieu de préférence dans une zone ombragée. Les femelles déposent leurs oeufs par petits paquets (3 à 7) sous l'épiderme du fruit-hôte, à 2-5 mm de profondeur. Après la ponte, la femelle dépose autour du point de piqûre une phéromone de marquage. La fécondité totale est d'environ 400-600 oeufs en conditions favorables.

Les oeufs éclosent après 2 à 5 jours. Les larves s'enfoncent alors dans la pulpe du fruit, où le cycle larvaire, qui comprend trois stades, dure de 9 à 15 jours. En fin de développement, les asticots quittent le fruit d'une brusque détente pour s'enfoncer à faible profondeur dans le sol, où s'effectue la nymphose. Celle-ci s'effectue pendant une durée variable selon les conditions climatiques (10-11 jours en été et 18-20 jours en automne pour l'Europe).

En conditions de température favorable, le cycle de développement complet dure d'une quinzaine à une vingtaine de jours (respectivement à 32 et 26 °C). Le seuil de développement se situe à 14 °C. Le nombre de générations annuelles varie en fonction des conditions climatiques. Les précisions sur le comportement de cet insecte nous ont permis de définir le modèle sous la forme DEVS.

6.2.2 Modèle de dispersion de mouches des fruits

Ce modèle utilise la technique des automates cellulaires, chaque cellule constituant un modèle atomique et contenant la dynamique du modèle. Le comportement du modèle est décrit directement

en code JAVA, l'interface générant le squelette du code contenant à la fois, les quatre fonctions et les ports situés aux points cardinaux.

Ce modèle ne décrit pas directement le comportement des mouches comme pourrait le faire un modèle multi-agents, mais plutôt le comportement d'une parcelle de verger contenant ces insectes. Chaque cellule est ici une zone de verger contenant une espèce unique repérée par sa date et sa durée de maturation. Chaque zone représentée par la cellule contient un certain nombre de mouches qui peuvent transiter vers les cellules voisines si les fruits contenus dans ces cellules sont mûrs. Les mouches contenues dans une zone sont caractérisées par un stade de maturité (Pupe, Larve, Adulte, Oeuf) et par une quantité ; une zone ne pouvant en effet contenir qu'une population de mouches et dans un seul état de maturité. Il existe une corrélation forte entre la température et le développement de la mouche, ce qui amène à calculer les dates de changement de stade des populations de mouches actives en utilisant le nombre de degrés/jour (somme des températures journalières). Le seul format des données de température disponible en entrée est journalier, cela pose des problèmes de conception du modèle dans le cadre d'une simulation à événements discrets car il est alors nécessaire de sommer les valeurs de température pour calculer l'évolution de la population. Pour les populations de mouches, le passage d'un état à l'état suivant est donc programmé immédiatement par la fonction d'avancement du temps lorsque le seuil de maturité (en degrés/jour) est atteint.

La figure 6.10 présente la structure interne utilisée par le modèle. Nous ne disposons en général que d'une seule mesure de température pour un domaine, le sous-modèle **distributeur** permet de répartir les mesures journalières de températures à toutes les cellules. Le sous-modèle **Domaine** est un modèle couplé de type domaine cellulaire contenant les cellules représentant les multiples zones du verger.

Une cellule possède la structure suivante :

$< X\{N,S,E,O,in\}, Y\{N,S,E,O,out\}, S\{$état-fruit, date-matur, dur-matur, état-mouche, quantité, djour$\}, \delta_{int}, \delta_{ext}, \lambda, t_a >$

où :

– `N, S, E, O` correspond aux ports du voisinage Nord, Sud, Est et Ouest,

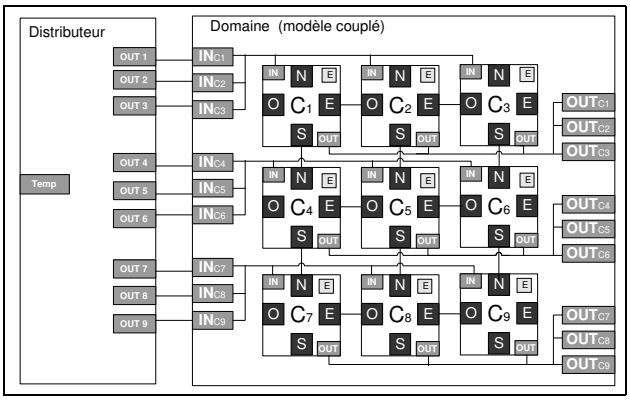

FIG. 6.10*: Structure interne du modèle de mouche*

– `in` est le port recevant la température du jour,
– `out` est le port émettant la quantité et l'état des mouches lorsque la population évolue,
– `état-fruit` est l'état des fruits contenus dans la cellule (mûrs, non-mûrs),
– `date-matur` est la date de maturité de l'espèce contenue dans la zone (en jour de l'année/365),
– `dur-matur` est la durée de maturation de l'espèce contenue dans la zone (en journées),
– `état-mouche` est l'état courant des mouches contenues dans la cellule (Pupe, Larve, Adulte, Oeuf),
– `quantité`, la quantité de mouches présentes dans la zone,
– `djour` est le niveau de maturité des mouches contenues dans la cellule (en degrés/jour).

Dans ce modèle :

– la fonction δ_{int} (interne) met à jour la quantité de degrés/jour et assure le changement d'état de la population des mouches en cas de dépassement de seuil. Dans le cas du passage Adulte vers Oeuf la population est multipliée par un coefficient de ponte ; pour tous les autres passages, la quantité présente est multipliée par un coefficient de survie. Cette fonction met aussi à jour le nombre de mouches contenues par la cellule en fonction des flux migratoires.
– La fonction δ_{ext} (entrée) reçoit la température du jour et la quantité de mouches.
– La fonction λ (sortie) envoie aux cellules voisines la quantité de mouches (forcément adultes) qui doivent quitter la zone lorsque les mouches doivent migrer.

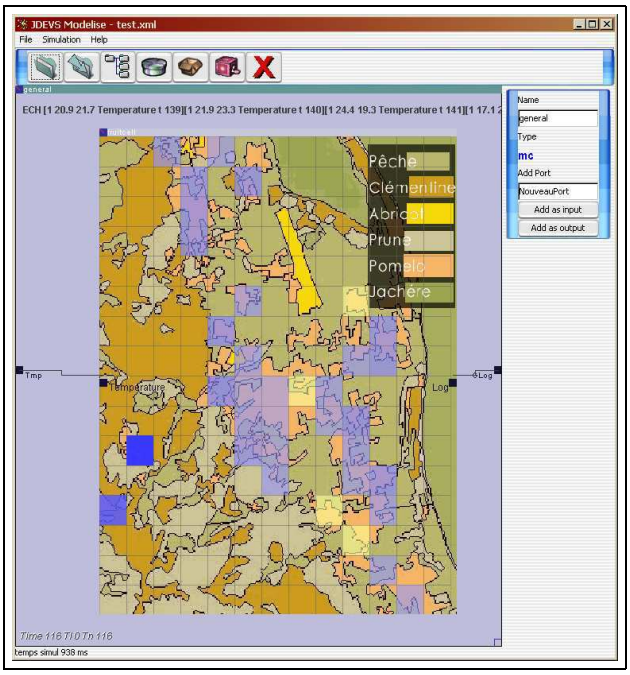

FIG. 6.11*: Modèle de mouches des fruits en cours de simulation dans JDEVS.*

– La fonction t_a (avancement du temps) définit le temps au prochain mûrissement des fruits en fonction des dates de maturation contenues dans cette cellule.

Une fois le comportement défini, il reste à faire le pré-traitement des données dans le SIG pour initialiser la simulation. En particulier les cartes de dates de maturation des fruits sont nécessaires. Pour chaque zone étudiée il est nécessaire de générer aussi une image de fond pour la simulation facilitant sa visualisation.

La simulation peut être directement lancée dans l'interface graphique de JDEVS comme présenté en figure 6.11 ; pour cela une palette est définie comprenant une couleur pour chaque état de la mouche, la concentration étant quant à elle, représentée par un niveau de transparence.

Le modèle est en cours de validation par le Service Régional de la Protection des Végétaux qui place actuellement des pièges à mouches pour comparer les concentrations réelles avec les concentrations simulées. Quelques résultats de simulation sont donnés dans la section suivante.

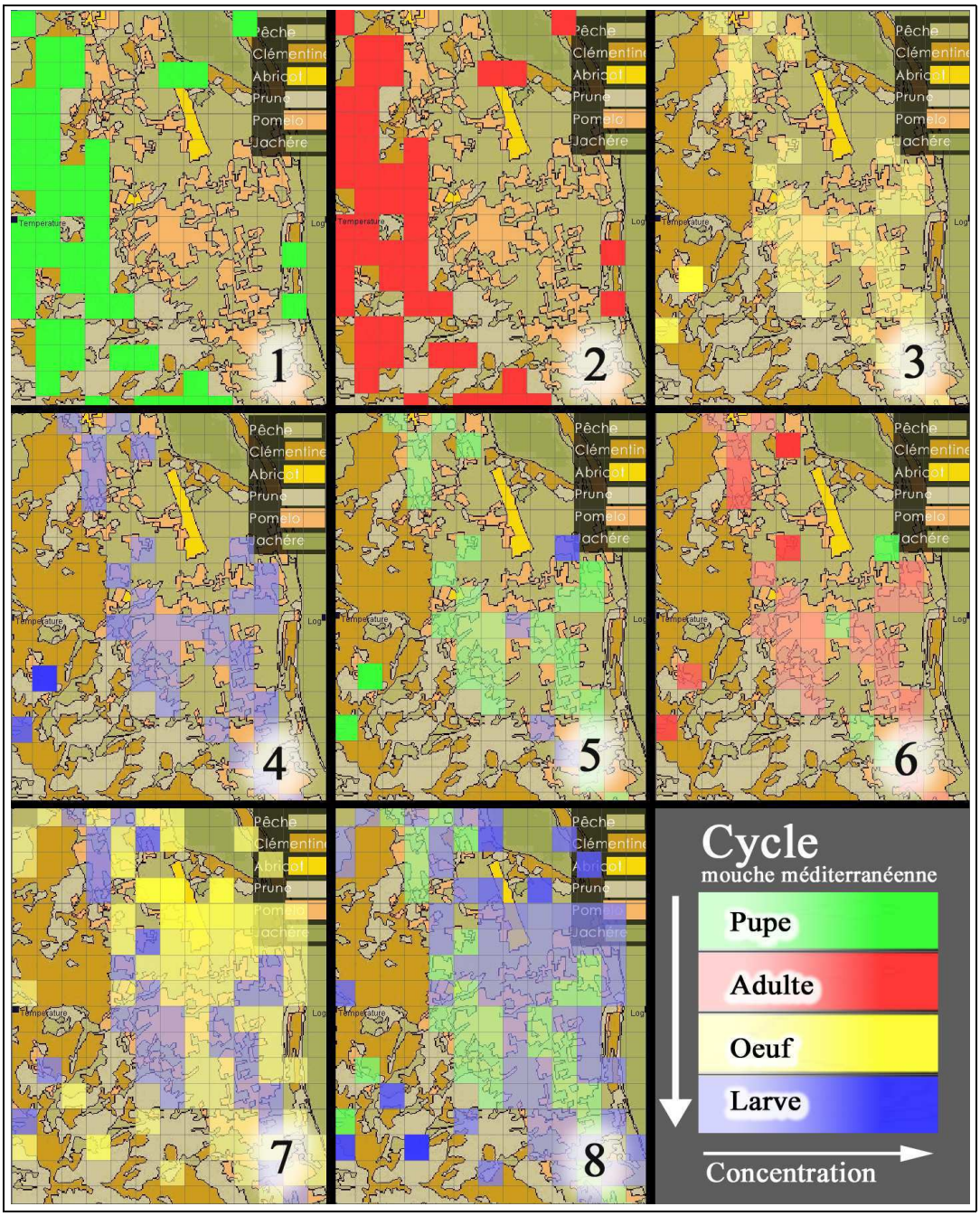

FIG. 6.12*: Résultats de simulation du modèle de calculs de dispersion des mouches des fruits.*

6.2.3 Résultats de simulation

La figure 6.12 présente le modèle de mouches des fruits pendant une simulation à l'intérieur de JDEVS. A l'origine, (figure 6.12.1) une population de mouches sous la forme de pupes est distribuée uniformément dans le champ de clémentiniers (espèce ayant la date de maturation la plus précoce). Ces pupes se transforment ensuite en adultes (figure 6.12.2) puis se dispersent rapidement dans le champ de pomelos avant de pondre (figure 6.12.3). Ces oeufs se transforment en larves (figure 6.12.4) et enfin pupes (figure 6.12.5), à l'exception de certaines cellules où l'arrivée des mouches adultes a été plus tardive. Une fois adultes (figure 6.12.6), les mouches peuvent envahir les champs d'abricotiers et de pêchers (figure 6.12.7) puis produire de nouvelles larves (figure 6.12.8).

La simulation montre que plus une zone contenant une espèce est isolée des autres zones contenant la même espèce, plus la concentration en mouches est forte. Cette tendance est confirmée par les études entomologistes 6.9, mais ne constituent pas le résultat principal de cette expérimentation. Le but de cette expérimentation est de montrer qu'il est facile de créer et de simuler des modèles cellulaires dans JDEVS. L'expérience suivante présente un modèle d'étude de polluants dans un bassin versant mettant ainsi en évidence, une autre facette du logiciel : le couplage entre Feedback-DEVS et les modèles cellulaires.

6.3 Polluants dans un bassin versant

Pour préserver des ressources naturelles comme l'eau, il est nécessaire de modéliser les phénomènes qui les altèrent. La problématique de l'eau est souvent traitée en se concentrant sur une bassin versant qui délimite la zone de confluence des eaux pour un exutoire. La gestion des bassins versants est l'un des thèmes majeurs de notre laboratoire, que nous avons d'abord traités par une approche à événements discrets dans [Aiello, 1997] [Delhom, 1997] puis en utilisant des réseaux de neurones dans [Chiari et al., 2000b], [Chiari et al., 2000a] et [Filippi, 2000]. Ces études nous ont montrées que les processus devant être modélisés sont complexes et nécessitent souvent l'usage de plusieurs techniques de modélisation différentes.

Cette section présente deux modèles distincts et indépendants traitant chacun d'une partie distincte de cette problématique. Nous verrons dans la dernière partie de cette section que ces modèles sont ensuite couplés à l'intérieur d'un multi-modèles, démontrant ainsi la flexibilité offerte par un environnement de modélisation basé sur DEVS.

6.3.1 Modèle cellulaire de propagation de polluants

Le modèle de propagation de polluants est une version adaptée et simplifiée du modèle disponible dans [Zeigler et al., 1996], il a été développé pour simuler le flux d'eaux polluées à l'intérieur d'un bassin versant. Ce modèle, volontairement simple, ne simule pas l'écoulement de l'eau après une pluie, mais plutôt estime le chemin que suivra un flux d'eau polluée arrivant sur une zone d'un domaine.

Le modèle utilise la technique de modélisation par automates cellulaires standard, chaque cellule étant définie par un modèle atomique. Comme tout modèle atomique, une cellule est définie dans un fichier unique, son comportement est directement spécifié en code Java. Ce fichier contient les quatre fonctions du modèle atomique ainsi que l'ensemble des états suivants :

$< X\{N,S,E,O,in\}, Y\{N,S,E,O,out\}, S\{poros, elev, pollut\} >$

Les ports `N`, `S`, `E` et `O` correspondent aux ports Nord, Sud, Est et Ouest connectés aux cellules du voisinage. Le port `in` est le port d'entrée d'eau polluée externe sur la cellule et le port `out` le débit d'eau polluée en sortie de cellule.

Dans ce modèle :

- La fonction λ (sortie) envoie aux cellules voisines son élévation et la quantité d'eaux polluées qu'elle contient. Elle est appelée par le simulateur en cas d'activation.
- La fonction δ_{ext} (entrée) reçoit la quantité d'eaux polluées et l'altitude des cellules voisines et envoie un message d'activation si son élévation est plus faible que celle des cellules voisines contenant des polluants.
- La fonction δ_{int} (interne) est appelée quand la cellule doit mettre à jour ses états internes (quantité d'eau polluée contenue) en fonction de la quantité reçue et de la porosité du sol.

– La fonction t_a (avancement du temps) définit le temps à la prochaine activation de la cellule en fonction de la quantité d'eau polluée, déterminant ainsi la vitesse d'écoulement

Présenter l'intégralité du code Java de ce modèle ne présente pas d'intérêt à l'intérieur de ce chapitre. La partiede code java suivante présente la transcription de la fonction de sortie, λ :

```
EventVector outFunction(Message m){
  e = new EventVector();
  e.add(new Event(N,"Elev","Pollut"));
  e.add(new Event(S,"Elev","Pollut"));
  e.add(new Event(E,"Elev","Pollut"));
  e.add(new Event(O,"Elev","Pollut"));
  e.add(new Event(out,"","Pollut"));
  return e;}
```

Implémenter ces quatre fonctions constitue la seule tâche que le spécialiste doit réaliser afin d'avoir un modèle fonctionnel.

Une fois le comportement de la cellule basique défini, il faut réaliser le pré-traitement de l'information dans le SIG pour permettre à l'environnement de récupérer les informations via le gestionnaire spatial (ici création des cartes raster : porosité, élévation et quantité de polluant initial pour une taille de cellule identique).

Le panel de simulation 3D sert de cadre expérimental à la simulation du modèle. Java3D est ici utilisé pour afficher les états de chaque cellule. La carte d'élévation permettant pour sa part de reconstituer un modèle numérique tridimensionnel du terrain.

Il est possible d'interagir avec le modèle en cliquant sur une cellule de la carte pour envoyer des messages prédéfinis (ici, une décharge d'eau polluée). La sortie générale de la simulation est un fichier contenant l'ensemble des événements ayant entraîné un changement d'état de la zone. Il est aussi possible de reconstituer des cartes au format matriciel de ces événements et de les exporter vers un SIG par le gestionnaire spatial.

L'étude de la pollution d'un bassin versant implique qu'il existe une zone de pollution à l'inté-

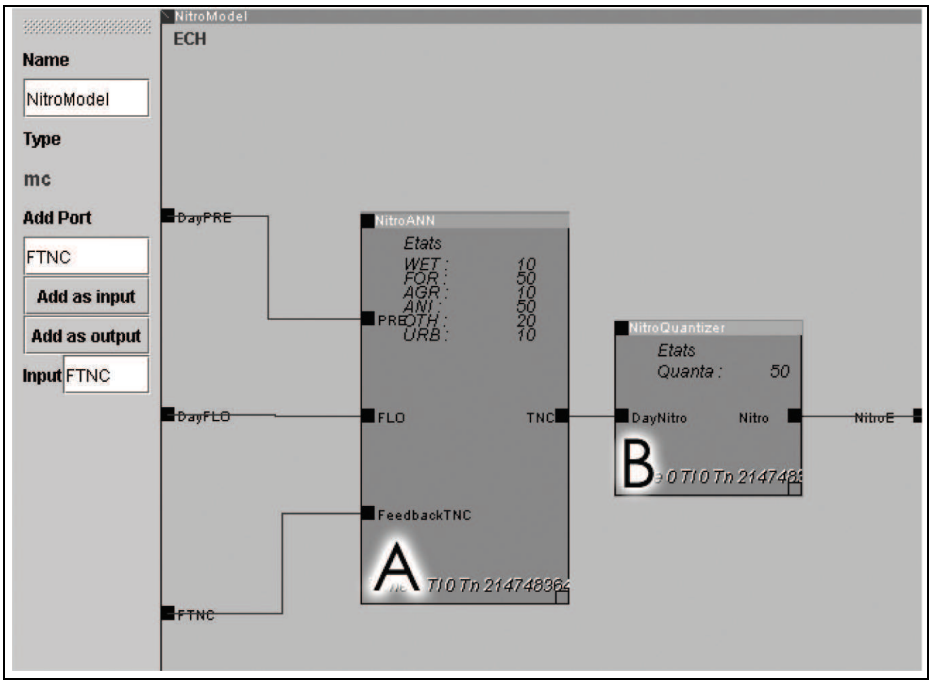

FIG. 6.13*: Modèle de calcul de concentration en nitrates dans JDEVS*

rieur de ce bassin. Ce modèle cellulaire fonctionne de manière autonome si l'utilisateur décide seul de la quantité de polluants à décharger dans une zone. Pourtant les données disponibles sur une zone sont rarement directement relatives à la quantité de polluants produits, mais plutôt au type de zone (surface urbanisée, forêt, agriculture et élevage). Le modèle suivant présente un modèle permettant d'estimer la quantité de polluants déchargés par une zone en fonction de son type.

6.3.2 Modèle de calcul de concentration en nitrates

Le modèle de calcul de concentration en nitrates est adapté de [Lek et al., 1999]. Ce modèle utilise Feedback-DEVS pour implémenter un réseau de neurones artificiels à l'intérieur d'un modèle DEVS à la manière du modèle de vieillissement de batterie présenté précédemment.

La Figure 6.13 représente le modèle à l'intérieur de l'interface de modélisation de JDEVS. Le modèle à réseau de neurones a été entraîné pour estimer la quantité journalière d'eau chargée en nitrates *(TNC)* rejetée par une zone.

Les variables définissant cette zone sont :

– *FOR*, Le pourcentage de forêt dans la zone

– *AGR*, le pourcentage de zones cultivées,

– *URB*, le pourcentage de zones urbaines,

– *WET*, le pourcentage de zones humides (marais, lacs),

– *OTH*, le pourcentage de zones d'un autre type,

– *ANI*, le nombre d'animaux.

Le modèle possède aussi deux ports d'entrée :

– `FLO`, l'écoulement journalier sur la zone (en litre),

– `PRE`, les précipitations journalières sur la zone (en litre).

Dans la figure 6.13, le composant **A** correspond au modèle feedback-DEVS pré-entraîné, implémentant le réseau de neurones.

Dans ce modèle les paramètres *FOR*, *AGR*, *URB*, *WET*, *OTH* et *ANI* permettent de définir la composition de la zone. Le modèle permet ainsi de simuler la quantité de nitrates émise par un terrain selon son type. Les entrées de ce modèle sont les précipitations et l'écoulement journalier, qui permettent de déterminer la quantité de nitrates émise puis transportée dans les zones voisines. La quantité journalière de nitrates est transmise par le port `TNC`.

Ce modèle possède aussi un port d'entrée de feedback, `FeedbackTNC` qui permet d'activer la fonction d'apprentissage d'une valeur de *TNC* pour la valeur courante contenue dans `FLO` et `PRE`.

Le principal avantage de ce modèle est la disponibilité des données d'apprentissage, de test et de validation. Le modèle se comporte de façon identique au modèle d'origine décrit dans [Lek et al., 1999]. Néanmoins ce modèle fonctionne en recevant des activations avec un pas de temps horaire, ce qui en fait un modèle dirigé par horloge. Pour permettre l'intégration de ce modèle dans une simulation DEVS il est nécessaire de rajouter un composant de sortie permettant l'intégration de ce modèle dans le contexte d'une simulation à événements discrets.

Le composant de sortie est un est un quantificateur, présenté dans la figure 6.13**B**. Le fonctionnement du quantificateur est simple : il somme les quantités d'eau chargée en nitrates reçues par le port d'entrée **DayNitro** et lorsque cette quantité dépasse le seuil défini par le paramètre *quanta* du modèle, un événement est émis par le port `Nitro`.

Pour initialiser la simulation, la série temporelle de précipitation et d'écoulement journalier

est chargée comme liste d'événements d'entrée. La sortie de la simulation n'est pas la quantité journalière de nitrates, mais une liste d'événements informant du temps de dépassement du seuil défini par le paramètre *quanta*. Le format de sortie, qui n'est pas dépendant d'une horloge, permet un couplage simplifié de ce modèle avec d'autres modèles utilisant une technique différente. La section suivante présente le couplage de ce modèle avec le modèle cellulaire de propagation de polluants.

6.3.3 Couplage du modèle de calcul de concentration en nitrates avec le modèle cellulaire de propagation de polluants

Le modèle de propagation de polluants est un modèle cellulaire et le modèle de concentration en nitrates est un modèle utilisant les réseaux de neurones, pourtant, grâce à l'utilisation de DEVS comme méthodologie de base, ils partagent tous deux des interfaces contenant des ports d'entrée et de sortie compatibles. Il est donc possible de coupler ces modèles et ainsi de simuler l'influence d'une nouvelle composition de zone sur un bassin versant.

Avant d'effectuer le couplage, le modélisateur doit vérifier si les données devant passer d'un modèle vers un autre, sont compatibles. Dans cette expérience, les données correspondent : le modèle de calcul de nitrates émettant une quantité d'eau chargée en nitrates et le modèle cellulaire recevant une quantité d'eau chargée en polluants.

Le modélisateur doit ensuite choisir une zone d'étude et collecter les données de précipitation et d'écoulement pour cette zone. Il est ensuite possible de tester l'impact de différente types de zone en modifiant les variables FOR, AGR, URB, WET, OTH et ANI.

Le couplage des deux modèles est fait par l'interface graphique de modélisation, en intégrant les deux sous-modèles dans un modèle couplé. Après avoir chargé le domaine à partir du SIG dans le modèle cellulaire, il est possible de choisir une cellule, puis le port de cette cellule que l'on souhaite lier et enfin établir la connexion avec la sortie générale du modèle de calcul de concentration de nitrates.

La balise XML résultant de la connexion est présentée en Figure 6.14 : elle définit une

```
<IC><LINK>
        <URI>./nitrogen.xml</URI><PORT>NitroE</PORT>
        <URI>./cell.xml</URI><PORT>in[40-83]</PORT>
</LINK></IC>
```

FIG. 6.14*: Balise XML de couplage entre deux modèles.*

connexion interne au modèle couplé entre le port de sortie `NitroE` du modèle de concentration en nitrogène et le port `in[40-83]` de la cellule choisie (ici la cellule se trouvant à la ligne 40 et colonne 83) .

La simulation peut ensuite être initialisée depuis l'interface graphique. Le chargement du modèle déclenche l'ouverture des interfaces de simulation associées à chacun des sous-modèles.

6.3.4 Résultats de simulation

La figure 6.15 présente un exemple de simulation ou une source de polluants à été couplée à une cellule située dans la partie haute d'un bassin versant. L'interface de visualisation 3D est utilisée ici pour permettre une meilleure visualisation de la propagation de polluants.

Dans cette section, nous avons présenté un exemple de modélisation d'un multi-modèles combinant un modèle réseaux de neurones et un modèle cellulaire. L'architecture générique sur laquelle est basée JDEVS permet le couplage simplifié de ces deux modèles utilisant des techniques différentes car les procédures de couplages de chacun de ces modèles sont conçues pour fonctionner avec des modèles hiérarchiques basés sur des composants.

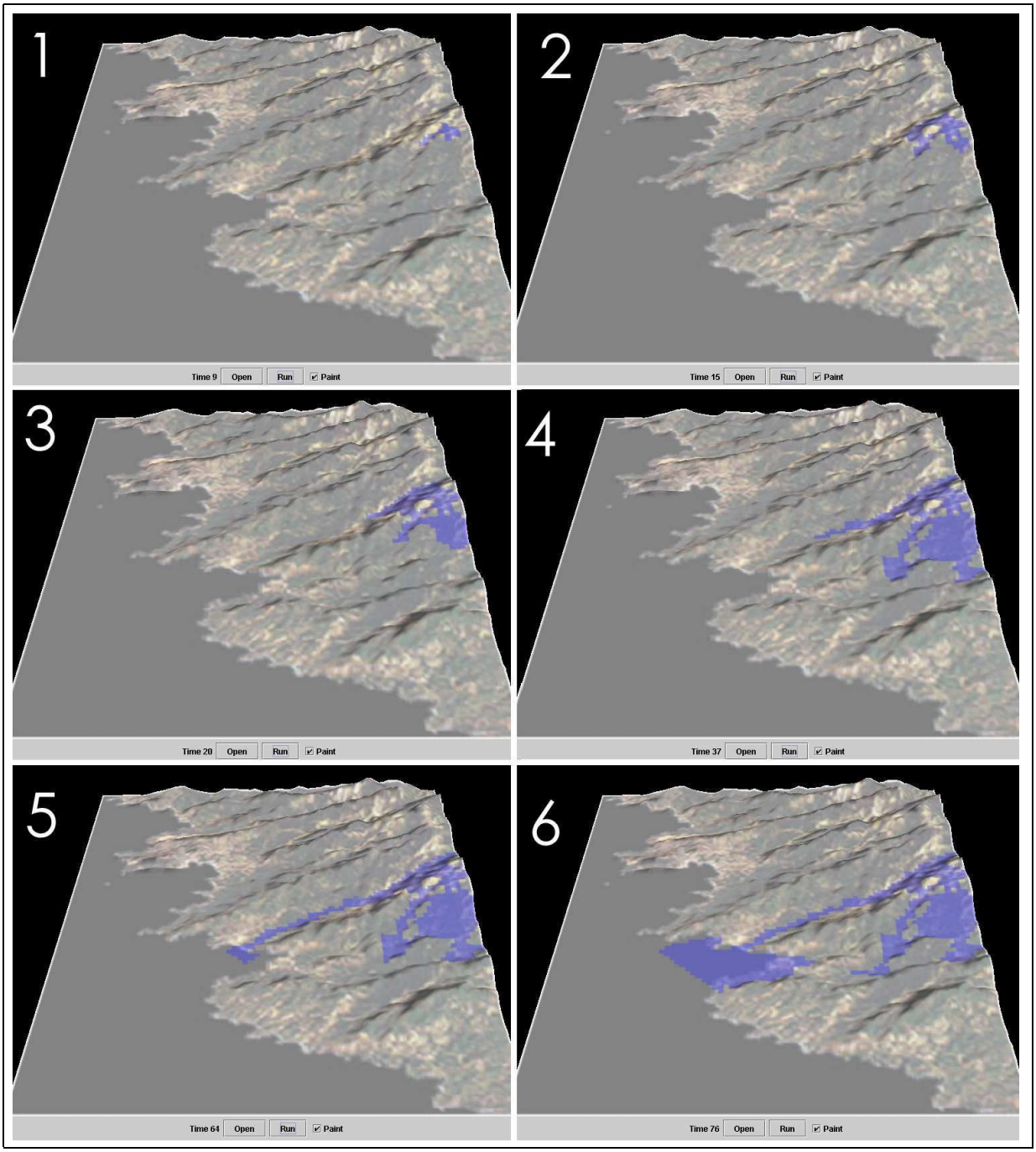

FIG. 6.15*: Vue de la simulation du modèle de propagation de polluants dans le sud-est de la Corse (Bassin du Taravo et du Rizzaneze)*

6.4 Conclusion

Nous avons illustré l'approche de modélisation avec JDEVS par trois modèles. Ainsi, nous avons pu démontrer la réalité du développement de multi-modèles, combinant des modéles alliant plusieurs techniques de modélisation de maniére simple et interactive.

Plusieurs autres études de systèmes ont utilisé JDEVS comme environnement de modélisation et de simulation, en particulier concernant la simulation de feux de forêts [Muzy et al., 2001] et la simulation d'architectures logicielles [de Gentili, 2002]. D'autres modèles utilisant Vector-DEVS sont en cours de développement, ce qui constitue principalement une de nos perspectives de recherche. Ces perspectives sont présentées après le bilan de nos travaux dans le chapitre suivant.

Conclusion générale

A TRAVERS ce document, nous avons présenté une approche logicielle permettant la création de modèles hétérogènes, ou multi-modèles. L'objectif final du travail était de disposer d'un outil robuste et évolutif de simulation destiné à l'étude de systèmes naturels complexes. Pour cela, nous avons analysé les approches existantes et fait le choix de baser cet outil sur un formalisme unificateur dans le domaine de la modélisation et de la simulation : DEVS. A partir de ce formalisme, nous avons proposé une architecture logicielle modulaire et orientée objets, permettant l'intégration et le couplage de méthodologies diverses d'analyses de systèmes par la simulation. La validation de cette architecture a été faite en deux temps, d'abord par l'ajout de nouvelles techniques de modélisation puis par une réalisation logicielle, JDEVS.

Ce dernier chapitre présente, dans sa première partie, un bilan des contributions de ce travail de thèse dans le domaine de la modélisation et la simulation de systèmes naturels complexes. La seconde partie est consacrée aux perspectives de recherche.

Bilan des travaux

Notre démarche a consisté, en premier lieu, à examiner les concepts régissant la définition et l'utilisation de modèles de systèmes complexes. La spécificité de tels types de modèles est qu'ils

sont composés par agrégation d'autres modèles pouvant être différents en nature. Pour assurer la cohérence de tels modèles, nous avons choisi une méthodologie de décomposition de système, la *modélisation multifacettes* [Zeigler, 2000].

Le formalisme DEVS a été adopté pour appliquer cette méthode. Nous avons montré sa pertinence comme base unificatrice permettant la coopération de différents types de modèles. Le développement formel d'un environnement basé sur cette méthodologie et le formalisme DEVS est le fruit de travaux développés depuis cinq ans dans notre laboratoire [Aiello, 1997], [Delhom, 1997]. Ces travaux constituent les bases de la définition de notre environnement logiciel.

Toutefois, ces bases se limitent à une présentation formelle d'un environnement orienté objet de modélisation et de simulation à événements discrets ; d'un point de vue plus conceptuel, nous nous sommes appuyés sur le principe de multi-modélisation. La multi-modélisation introduit les concepts qui autorisent le couplage de modèles utilisant des techniques différentes (automates cellulaires, multi-agents, ..) grâce à l'utilisation d'interfaces d'entrées/sorties bien définies.

Les modalités d'intégration de différentes techniques de modélisation de systèmes ont donc constitué les enjeux majeurs de notre analyse.

Afin de positionner plus clairement notre travail, nous avons ensuite effectué une revue représentative des approches logicielles de modélisation de systèmes complexes existants. La premiere partie de cette revue concernait les environnements basés sur DEVS. Cette partie nous a amené à identifier les principales caractéristiques d'intégration logicielles de ce formalisme. Parmi ces caractéristiques nous avons relevé la décomposition en plusieurs modules (interface graphique, moteur de simulation, bibliothèque) ainsi que les normes, existantes ou en cours de développement que doivent respecter de tels environnements (HLA (High Level Architecture) [Zeigler et al., 1999] ou le standard DEVS [DEVS-Group, 2003]).

La seconde partie de cette revue à mis en relief les principales singularités des logiciels d'étude de systèmes naturels. Cette revue nous a notamment permis d'identifier les besoins logiciels en terme d'entrée/sortie, de stockage, d'architecture et de modalités d'expérimentations. Ces singularités, ainsi que les besoins spécifiques liés à l'utilisation de DEVS ont constitué la base pour l'architecture logicielle que nous avons développée.

Le cadriciel ainsi élaboré est basé sur la notion de packages. En effet le principal objectif était que ce cadriciel soit suffisamment ouvert et tolère l'intégration de nouvelles techniques de modélisation en gardant la base intacte.

Pour ce faire nous avons proposé une séparation explicite entre les packages principaux et les packages des techniques de modélisation et de simulation. Les packages principaux sont constitués de l'interface de modélisation, du moteur de stockage, des cadres expérimentaux et du package comprenant les classes qui implémentent le formalisme DEVS. La décomposition en packages relativement indépendants est ici gage d'évolutivité car elle permet d'améliorer chaque partie sans mettre en défaut la cohérence de l'ensemble. La décomposition en quatre packages facilite l'intégration de nouvelles techniques en offrant une séparation claire des fonctions entre chaque partie à implémenter.

Pour en arriver au cadriciel final, plusieurs acteurs ont été associés au processus développement, le monde du logiciel libre à travers la publication du logiciels sur internet [Freshmeat, 2002], les spécialistes d'études de systèmes naturels à travers des programmes de recherches [Filippi et al., 2001], [Faure, 2001] et la communauté scientifique par de nombreuses publications.

Nous avons ensuite proposé l'ajout de trois techniques supplémentaires de modélisation et de simulation. La première, Feedback-DEVS, se révèle utile à la spécification de modèles adaptatifs, auto-apprenants, souvent utilisés pour construire des modèles empiriques. L'intégration de cette technique a été facilitée par sa similarité avec la technique DEVS classique déjà présente dans le cadriciel.

La seconde technique basée sur les automates cellulaires, est la méthode la plus fréquemment utilisée pour l'étude de systèmes ayant des dimensions spatiales. Pour rendre cette technique disponible, nous avons du ajouter des modes de représentation de l'espace à notre outils et développer les cadres expérimentaux et les interfaces vers des outils d'analyse spatiales (SIG).

Enfin, la troisième technique ajoutée, Vector-DEVS est une approche originale adaptée des méthodes mathématiques de "front tracking" [Glimm et al., 1996] permettant l'étude de la propagation d'un phénomène par la simulation de la dynamique de son interface physique. Pour intégrer

cette approche nous avons cherché à appliquer le formalisme DSDEVS [Barros, 1997] pour qu'il prenne en compte la gestion de l'espace. Ce formalisme a été choisi car il autorise la définition de modèles à structure dynamique. Le côté spatial a été introduit par la définition des concepts d'*agent géographique* (modèle atomique géo-référencé) et de *gestionnaire de forme* (réseau DSDEVS intégrant une dynamique structurelle et spatiale). La spécification formelle Vector-DEVS a été présentée et son ajout dans le cadriciel effectué d'un point de vue conception orientée objet. L'intégration logicielle de cette technique est en cours de réalisation et constitue une des principales perspectives de recherches.

Une implémentation du cadriciel a ensuite été proposée. JDEVS, largement utilisé dans notre laboratoire, permet la modélisation et le couplage de multi-modèles ainsi que leur stockage au format XML et leur publication sur le web.

Enfin, nous avons illustré l'approche de modélisation avec JDEVS par la création de trois modèles développés en étroite collaboration avec des spécialistes : un multi-modèle de système photovoltaïque intégrant une batterie modélisée grâce à un réseau de neurones artificiel [Notton et al., 2002], un modèle cellulaire de propagation de mouches des fruits méditerranéennes [Faure, 2001] et un multi-modèle de propagation de polluants couplant modélisation cellulaire et réseau de neurones [Chiari et al., 2000a]. Ces expérimentations ont permit de tester et de valider le logiciel JDEVS et consequemment le cadriciel.

L'ensemble de ces travaux a fait l'objet de publications parmi lesquelles nous pouvons citer : [Filippi et Bisambiglia, 2003], [Filippi, 2002], [Filippi et al., 2001], [Filippi et al., 2002b], [Filippi et Bisgambiglia, 2002], [Bernardi et al., 2003] et [Filippi et al., 2002a].

Perspectives de travail

Plusieurs perspectives de recherches et de réalisations se dessinent à l'issue de ce travail. Dans un premier temps nous devons réaliser une étude de système à l'aide de Vector-DEVS, ce qui permettra de terminer l'implémentation de cette technique ainsi que son intégration dans le cadriciel. Il est prévu de réaliser des simulations comparatives entre automates cellulaires et Vector-DEVS pour mieux cerner les avantages et inconvénients de chaque méthode. Nous projetons par ailleurs l'ajout d'autres méthode de décomposition d'évolution de front, en particulier la méthode de *mouvement par courbure moyenne* [Buckdahn et al., 1988]. En effet, seul le principe de décomposition des ondes de Huygens est actuellement disponible, ce qui limite le champ d'application de la technique aux systèmes de diffusion.

Nous envisageons également de poursuivre la démarche d'évolution de l'outil en faisant plus d'études de cas, ce qui nous amènera à construire davantage de modèles et ainsi enrichir notre bibliothèque. A ce titre nous comptons profiter des efforts de standardisation du formalisme DEVS pour partager un format de stockage commun avec les autres outils du domaine. A plus court terme, nous pensons réaliser une connexion entre JDEVS et Atom3 [Vangheluwe, 2000] pour bénéficier de la transformation automatique des différents formalismes en DEVS.

A moyen terme, nous comptons modifier notre outil pour qu'il puisse adopter la norme HLA [Zeigler et al., 1999] qui définit les protocoles de communications entre modèles en cours de simulation. Actuellement notre outil n'autorise pas l'interconnexion avec d'autres logiciels. Par contre, nous bénéficions d'une architecture modulaire nous autorisant à intégrer un moteur de modélisation et de simulation DEVS standard respectant cette norme. Ce module fait actuellement l'objet d'une recherche partagée entre les différents instituts travaillant sur ce formalisme [DEVS-Group, 2003].

D'autres techniques devront aussi être intégrées à l'outil pour qu'il puisse prétendre devenir une plate-forme généraliste de modélisation de systèmes naturels complexes ; nous pensons en particulier à la modélisation individu-centré [Duboz et al., 2003] et à la modélisation de processus stochastiques.

La vitesse de simulation est une priorité à plus long terme, pour cela, il nous faudra substituer

au moteur DEVS le moteur Parallel-DEVS [Zeigler, 2000] pour ainsi bénéficier de possibilités de simulation sur machines parallèles ou distribuées [Gimblett et al., 1995].

Il nous faut travailler sur des interfaces de visualisation et de capture de données plus adaptées à l'étude de systèmes naturels. En ce qui concerne la visualisation de la simulation, plusieurs utilisateurs ont exprimé le besoin d'outils d'études statistiques autorisant la visualisation du comportement du modèle en cours de simulation. Le décryptage de la liste d'événements de sortie est en effet très difficile sans outils d'analyse tiers et ne peut s'effectuer qu'une fois la simulation terminée. Concernant la capture de données, nous réfléchissons à la définition de capteurs intelligents permettant de communiquer directement par des messages DEVS avec un modèle. De tels capteurs pourraient par exemple servir à envoyer des données de vent, de pression, chaleur ou autres paramètres à des modèles dans le cadre d'une simulation temps réel.

Au delà des considérations techniques et scientifiques, nous espérons que le logiciel, une fois achevé, sera utilisé comme un véritable outil d'aide à la décision par les responsables en charge du développement et de la préservation du patrimoine naturel régional, aérien, terrien et marin.

Liste des publications

Publications réalisées dans le cadre de la thèse

Article dans des journaux internationaux avec comités de lectures

– Filippi, J-B. et Bisambiglia, P. (2003). *JDEVS :an implementation of a DEVS based formal framework for environmental modelling*. Environmental modelling and Software journal. A paraître.

Conférences internationale avec comité de lecture et publication des actes

– Bernardi, F., Filippi, J-B., et Santucci, J. (2003). *A generic framework for environmental modeling and simulation*. Dans Actes de la conférence IEEE International Conference on Systems, Man and Cybernetics. pages 1810-1815, Washington, USA.

– Filippi, J-B. et Bisgambiglia, P. (2002). *Enabling large scale and high definition simulation of natural systems with vector models and JDEVS*. Dans Actes de 2002 IEEE, SCS, ACM, Winter Simulation Conference, pages 1964-1970. San Diego, CA, USA.

– Filippi, J-B., Bernardi, F., et Delhom, M. (2002). *The JDEVS environmental modeling and simulation environment*. Actes de la conférence IEMSS 2002 conference on Integrated Assessment and Decision Support, 3 :283-288. Lugano, Suisse.

– Filippi, J-B., Chiari, F., et Bisgambiglia, P. (2002). *Using JDEVS for the modeling and simu-*

lation of natural complex systems. Dans Actes de la conférence SCS AIS 2002 Conference, volume 1, page 317-322. Lisbonne, Portugal.

– Filippi, J-B., Bisgambiglia, P., et Delhom, M. (2001). *Neuro-devs, an hybrid methodology to describe complex systems*. Dans Actes de SCS ESS 2001 conference on simulation in industry, volume 1, pages 647-652. Marseille, France.

– Chiari, F., Delhom, M., Filippi, J-B., et Santucci, J. (2000). *Prédiction du comportement hydrologique d'un bassin versant à l'aide de réseau de neurones.* Dans Actes de la conférence ESRI France SIG 2000. CD-ROM, Paris, France.

– Chiari, F., Delhom, M., Santucci, J., et Filippi, J-B. (2000). *Prediction of the hydrologic behavior of a watershed using artificial neural networks and geographic information systems.* Dans Actes de la conférence IEEE International Conference on Systems, Man and Cybernetics. CD-ROM, Nashville, TS, USA.

Conférences internationale sans comité de lecture

– Filippi, J-B. (2002). *An hybrid method, combining neural networks, object oriented modeling and GIS for the simulation of natural complex systems*. Dans Actes du congrès GISIG "eurogeowater". CD-ROM, Oxford, GB.

– Bernardi, F., Filippi, J-B., et Santucci, J. (2001). *Xml object-oriented models libraries with web-based access capacities*. Dans Actes de la conférence ICSSEA, Génie logiciel et ingénierie de systèmes. Paris, France.

Rapports de recherche

– Filippi, J-B. (2000). *Analyse, conception et programmation d'un logiciel de simulation de bassins versants*. Rapport technique, University of Corsica. TCUDC5092000.

Posters

– Filippi, J-B. (2002). *JDEVS pour la modélisation et simulation orientée objet de systèmes naturels*, Corté, France. Poster, congrès environnement et identités en mediterranée.

– Filippi, J-B. (2002). *Large scale spatial simulation with Vector-DEVS*. Poster, Winter Simulation Conference, San-Diego, USA.

Publications réalisées dans le cadre de contrats de recherches

Conférences internationale avec comité de lecture et publication des actes

– Barthélémy, S. et Filippi, J-B. (2003). *A typology of very small companies using self-organizing maps*. Dans Actes de la conférence IEEE International Conference on Systems, Man and Cybernetics. Pages 3518-3523, Washington, USA.

Rapports de recherche

– Apoteker, T., Barthélémy, S., Delhom, M., Filippi, J-B., Levratto, N., Mahérault, L., Revest, V., Rivaud-Danset, D., et Santucci, J. (2000). *Rapport intermédiaire. L'évaluation des entreprises afin de faciliter l'accès au crédit : quelle intermediation informationnelle.* Rapport technique, Ministère de l'Économie des Finances et de l'Industrie, Secrétariat d'état aux PME - Direction des Entreprises du Commerce, de l'Artisanat et des Services. Effectuée par TAC sous la supervision de N. Levratto (CNRS).

– Apoteker, T., Barthélémy, S., Delhom, M., Filippi, J-B., Levratto, N., Mahérault, L., Revest, V., Rivaud-Danset, D., et Santucci, J. (2001). *L'évaluation des entreprises afin de faciliter l'accès au crédit : quelle intermediation informationnelle.* Rapport technique, Ministère de l'Économie des Finances et de l'Industrie, Secrétariat d'état aux PME - Direction des Entreprises du Commerce, de l'Artisanat et des Services. Effectuée par TAC sous la supervision de N. Levratto (CNRS).

– Barthélémy, S., Delhom, M., Filippi, J-B., Levratto, N., et Mahérault, L. (2002). *Conditions de l'élaboration d'une base de données qualitatives sur les entreprises aux fins d'une intermediations informationnelle*. Rapport technique, Ministère de l'Économie des Finances et de l'Industrie, Secrétariat d'état aux PME - Direction des Entreprises du Commerce, de l'Artisanat et des Services.

– Barthélémy, S. et Filippi, J-B. (2003a). *Typologie d'entreprise en utilisant des cartes auto-organisatrices*. Rapport interne, Ministère de l'Économie des Finances et de l'Industrie.

Bibliographie

[Aiello, 1997] Aiello, A. (1997). *Environnement Orienté Objet de Modélisation et de Simulation à Evénements Discrets de Systèmes Complexes*. Thèse de Doctorat, Université de Corse.

[Arena, 2003] Arena (2003). Arena software. http ://www.arenasimulation.com/.

[Aumiaux, 2000] Aumiaux, M. (2000). *Initiation au langage VHDL, 2e édition*. Dunod. ISBN 2100045555.

[Barros, 1997] Barros, F. (1997). Modeling formalism for dynamic structure discrete event systems. *ACM Transactionson modeling and Computer Simulation*, 7 :501–515.

[Barros, 1998] Barros, F. (1998). Abstract simulators for the dsde formalism. *Actes de la conférence, 30th Winter simulation Conference*, pages 407–412. ISBN :0-7803-5134-7.

[Bernardi, 2002] Bernardi, F. (2002). *Conception de bibliothèques hiérarchisées de modèles réutilisables selon une approche orientée objet*. Thèse de Doctorat, Université de Corse.

[Bernardi et al., 2001] Bernardi, F., De-Gentili, E., et Santucci, J. (2001). Reusable models integration in a DEVS-based modeling and simulation environment. *Actes de la conférence SCS ESS 2001 simulation in industry*, 1 :644.

[Bernardi et al., 2003] Bernardi, F., Filippi, J.-B., et Santucci, J. (2003). A generic framework for environmental modeling and simulation. Dans *Actes de la conférence IEEE International Conference on Systems, Man and Cybernetics*, pages 1810–1815. Washington, USA.

[Bernardi et Santucci, 2002] Bernardi, F. et Santucci, J. (2002). Model design using hierarchical web-based libraries. Dans *Actes de la conférence 39th conference on Design automation*, volume 1, pages 14–17. New Orleans, USA.

[Bousquet et al., 1998] Bousquet, F., Bakam, I., Proton, H., et Le Page, C. (1998). Cormas : common-pool resources and multi- agent systems. *Lecture Notes in Artificial Intelligence 1416*, pages 826–838.

[Brandmeyer et Karimi, 2000] Brandmeyer, J. et Karimi, H. (2000). Coupling methodologies for environmental models. *Environmental Modelling and Software*, 15(5) :479–488.

[Buckdahn et al., 1988] Buckdahn, R., Cardaliaguet, P., et Quincampoix, M. (1988). A representation formula for the mean curvature motion. *SIAM Journal on Mathematical Analysis*, 33(4) :827–846.

[Campos, 2000] Campos, A. (2000). *Une architecture logicielle pour le développement de simulations visuelles et interactives individu-centrées*. Thèse de Doctorat, Université Blaise Pascal, Clermont-Ferrand.

[Capocchi et al., 2003] Capocchi, L., Bernardi, F., et Santucci, J. (2003). Transformation of vhdl descriptions into DEVS models for fault modeling. Dans *Actes de la conférence 2003 IEEE International Conference on Systems, Man and Cybernetics*. CD-ROM, Washington, USA.

[Carling, 1992] Carling, A. (1992). *Introducing neural networks*. Sigma Press, New York.

[Chiari et al., 2000a] Chiari, F., Delhom, M., Filippi, J.-B., et Santucci, J. (2000a). Prédiction du comportement hydrologique d'un bassin versant à l'aide de réseau de neurones. Dans *Actes de la conférence ESRI France SIG 2000*. CD-ROM, Paris, France.

[Chiari et al., 2000b] Chiari, F., Delhom, M., Santucci, J., et Filippi, J.-B. (2000b). Prediction of the hydrologic behavior of a watershed using artificial neural networks and geographic infor-

mation systems. Dans *Actes de la conférence IEEE International Conference on Systems, Man and Cybernetics*. CD-ROM, Nashville, TS, USA.

[Cincom, 2003] Cincom (2003). Cincom visualworks. http ://www.cincom.com/.

[Clavel et al., 1997] Clavel, G., Lopez, N., et Veillon, L. (1997). *Programmer objets avec Smalltalk*. Dunod. ISBN 2225851573.

[Coquillard et Hill, 1997] Coquillard, P. et Hill, D. (1997). *Modélisation et Simulation d'Ecosystèmes. Des modèles déterministes aux Simulations à événements discrets*. Masson, Paris. ISBN 2-225-85363-0.

[Dafermos, 1972] Dafermos, C. M. (1972). Polygonal approximations of solutions of the initial value problem for a conservation law. *J. Math. Anal. Appl*, 38 :33–41.

[Dahl et Nygaard, 1966] Dahl, O. J. et Nygaard, K. (1966). Simula - an algol based simulation language. *Transaction of the ACM*, 9(9) :671–678.

[Dauphin-Tanguy, 2003] Dauphin-Tanguy, G. (2003). *Les bond graph*. Modélisation objet avec UML. ISBN : 2212113978.

[de Gentili, 2002] de Gentili, E. (2002). *Contribution au développement d'une méthode de validation d'architecture logicielle de télécommunication à l'aide de la modélisation et la simulation à événements discrets*. Thèse de Doctorat, Université de Corse.

[Delhom, 1997] Delhom, M. (1997). *Modélisation et Simulation Orientées Objet, Contribution à l'Etude du Comportement Hydrologique d'un Bassin Versant*. Thèse de Doctorat, Université de Corse.

[Delhom et al., 1995] Delhom, M., Bisgambiglia, P., Santucci, J., et Aiello, A. (1995). Modeling and simulation of discrete events systems : the study of the hydrologic behavior of a catchment basin. Dans *Actes de la conférence IEEE International Conference on Systems, Man and Cybernetics*, volume 1, pages 19–30.

[DEVS-Group, 2003] DEVS-Group (2003). Groupe de standardisation de DEVS. http ://www.sce.carleton.ca/faculty/wainer/standard/.

[Diaz, 2001] Diaz, M. (2001). *Les réseaux de Petri, modèles fondamentaux*. Hermes Sciences Publication. ISBN : 2746202506.

[Duboz et al., 2003] Duboz, R., Amblard, F., Ramat, E., Deffuant, G., et Preux, P. (2003). Utiliser les modèles indidividus-centrés comme laboratoires virtuels pour identifier les paramètres d'un modèle agrégé. *Actes de la conférence MOSIM'03*. CD-ROM.

[Dukelow, 1994] Dukelow, J. (1994). Verification and validation of neural networks. Dans *Actes de la conférence Neural Network Workshop for the Hanford Community*, pages 76–80. Richland, Washington, USA.

[Faure, 2001] Faure, X. (2001). Modélisation et simulation de la dispersion de la mouche méditerranéenne des fruits (ceratis capitata) en corse. Rapport Technique 182, UMR CNRS 6134 Lab.

[Filippi et al., 2001] Filippi, J., Bisgambiglia, P., et Delhom, M. (2001). Neuro-DEVS, an hybrid methodology to describe complex systems. Dans *Actes de SCS ESS 2001 conference on simulation in industry*, volume 1, pages 647–652. Marseille, France.

[Filippi, 2000] Filippi, J.-B. (2000). Analyse, conception et programmation d'un logiciel de simulation de bassins versants. Rapport technique, University of Corsica. TCUDC5092000.

[Filippi, 2002] Filippi, J.-B. (2002). An hybrid method, combining neural networks, object oriented modeling and gis for the simulation of natural complex systems. Dans *Actes du congrès GISIG "euro-geowater"*. CD-ROM, Oxford, GB.

[Filippi, 2003] Filippi, J.-B. (2003). JDEVS, site web du projét. http ://spe.univ-corse.fr/filippiweb/.

[Filippi et al., 2002a] Filippi, J.-B., Bernardi, F., et Delhom, M. (2002a). The JDEVS environmental modeling and simulation environment. *Actes de la conférence IEMSS 2002 conference on Integrated Assessment and Decision Support*, 3 :283–288. Lugano, Suisse.

[Filippi et Bisambiglia, 2003] Filippi, J.-B. et Bisambiglia, P. (2003). JDEVS :an implementation of a DEVS based formal framework for environmental modelling. *Environmental modelling and Software*. A paraître, disponible sur http ://www.sciencedirect.com.

[Filippi et Bisgambiglia, 2002] Filippi, J.-B. et Bisgambiglia, P. (2002). Enabling large scale and high definition simulation of natural systems with vector models and JDEVS. Dans *Actes de 2002 IEEE, SCS, ACM Winter Simulation Conference*, pages 1964–1970. San Diego, CA, USA.

[Filippi et al., 2002b] Filippi, J.-B., Chiari, F., et Bisgambiglia, P. (2002b). Using JDEVS for the modeling and simulation of natural complex systems. Dans *Actes de la conférence AIS 2002 Conference*, volume 1, page 317. Lisboa, Portugal.

[Finney et Andrews, 1994] Finney, M. et Andrews, P. (1994). The farsite fire area simulator : Fire management applications and lessons of summer 1994. Dans *Actes de la conférence 1994 Interior West Fire Council Meeting and Program*, pages 209–216. Coeur Alene, USA.

[Fishwick, 1995] Fishwick, P. (1995). *Simulation Model Design and Execution : Building Digital Worlds*. Prentice Hall, nj, englewood cliffs edition.

[Fishwick, 1998] Fishwick, P. (1998). An architectural dsign for digital objects. *Actes de la conférence 1998 Winter Simulation Conference*, 1(1) :360–365.

[Freigassner et al., 2000] Freigassner, R., Praehofer, H., et Zeigler, B. (2000). Systems approach to validation of simulation models. *Cybernetics and Systems*, 1 :52–57.

[Freshmeat, 2002] Freshmeat (2002). JDEVS freshmeat project page. http ://freshmeat.net/jdevs.

[Frick, 2000] Frick, A. (2000). Multimodels : From simulation to object-oriented analysis and design. *16th IMACS World Congress 2000*, pages 21–25,. CD-ROM.

[FSERC, 2003] FSERC (2003). Florida solar energy center. http ://www.fsec.ucf.edu/.

[Gamma et al., sley] Gamma, E., Helm, R., Johnson, R., et Vlissides, J. (Addison- Wesley). *Design Patterns : Elements of Reusable Object Oriented Software*. Addison- Wesley. ISBN : 0-201-63361-2.

[Geotools, 2003] Geotools (2003). Geotools groupe d'utilisateurs. http ://www.geotools.org.

[Giambiasi et al., 2002] Giambiasi, N., Escudé, B., et Ghosh, S. (2002). Generalized discrete event simulation of dynamic systems. *SCS Transactions, Special Issue of Recent Advances in DEVS Methodology*.

[Gimblett et al., 1995] Gimblett, R., Ball, G., Lopes, V., Zeigler, B., Sanders, B., et Marefat, M. (1995). Massively parallel simulations of complex, large scale, high resolution ecosystem models. *Complexity International*, 2. ISSN 1320-0682.

[Glimm et al., 1996] Glimm, J., Simanca, S., Smith, T., et Tangerman, F. (1996). Computational physics meet computational geometry. Rapport Technique SUNYSB-96-19, State University of New York at Stony Brook.

[Halupka et al., 2000] Halupka, C., Bisgambiglia, P., et Santucci, J. (2000). Devs-based modelling of a pure thermal system. Dans *Actes de la conférence WMC 2000, San Diego, California*.

[Haykin, 1994] Haykin, S. (1994). *Neural Networks : A Comprehensive Foundation*. Macmillan.

[Hecht-Nielsen, 1989] Hecht-Nielsen, R. (1989). Neural network primer : part i. *AI Expert*.

[Hill, 1996] Hill, D. (1996). *Objetct-Oriented Analysis and Simulation.* Addison-Wesley Longmann. ISBN 0-201- 87759-7.

[Hill et al., 2000] Hill, D., Coquillard, P., Garcia, B., Traore, M., Mazel, C., et Campos, A. (2000). Multimodeling and object-oriented design patterns, application to bio-control simulation. *Actes de la conférence Artificial Intelligence Simulation (AIS 2000) Toward Collaborative/Distributed Modeling and Simulation Environments*, pages 219–228.

[Hopkins et Fishwick, 2003] Hopkins, J. et Fishwick, P. (2003). The rube framework for software modeling and customized 3-d visualization. *Visual Languages and Computing*, 1(14) :97–117.

[Hubbard, 1999] Hubbard, J. (1999). *Equations différentielles et systèmes dynamiques*. Vuibert. ISBN : 284225015X.

[Ilachinski, 2001] Ilachinski, A. (2001). *CELLULAR AUTOMATA, A Discrete Universe*. World Scientific Publishing Co. ISBN 981-02-4623-4.

[Johnson et Foote, 1988] Johnson, R. et Foote, B. (1988). Designing reusable classes. *Journal of Object-Oriented Programming*, 1-2 :22–31.

[Jungst et al., 2000] Jungst, R., Urbina, A., et Paez, T. (2000). Stochastic modeling of rechargeable battery life in a photovoltaic power system. Rapport Technique 1541C, Sandia national laboratories.

[Kofman et al., 2000] Kofman, E., Giambiasi, N., et Junco, S. (2000). Fdevs : A general DEVS-based formalism for fault modeling and simulation. Dans *Actes de la conférence 2000 European Simulation Symposium*, volume 1, pages 77–82. Hamburg, Germany.

[Kofman et S., 2001] Kofman, E. et S., J. (2001). Quantized state systems. A DEVS approach for continuous systems simulation. *Transactions of SCS*, 18(3) :123–132.

[Kwon et al., 1996] Kwon, Y., Park, H., Jung, S., et Kim, T. (1996). Fuzzy-DEVS formalism : Concepts, realization and applications. *Actes de la conférence AIS 1996 Conference*, pages 227–234.

[Lek et al., 1999] Lek, S., Guiresse, M., et Giraudel, J. (1999). Predicting stream nitrogen concentration from watershed features using neural networks. *International Journal an Geographical Information Systems*, 33(16) :3469–3478.

[Madonna, 2003] Madonna (2003). Berkeley madonna. http ://www.berkeleymadonna.com/.

[Mathworks, 2003] Mathworks (2003). Mathlab mathworks. http ://www.mathworks.com/.

[Means et Harold, 2001] Means, W. et Harold, E. (2001). *XML in a Nutshell*. O Reilly. ISBN : 0-596-00058-8.

[Minsky, 1968] Minsky, M. (1968). *Matter, minds and models*. Marvin Minsky, Ed.

[MODELICA, 2002] MODELICA (2002). Modelica user group. http ://www.modelica.org/.

[Muller et Gaertner, 1970] Muller, P. et Gaertner, N. (1970). *Modélisation objet avec UML*. Hermes Sciences Publication. ISBN : 2746201585.

[Muzy et al., 2001] Muzy, A., Marcelli, T., Aiello, A., Santoni, P., Santucci, J., et Balbi, J. (2001). An object oriented environment applied to a semi-physical model of fire spread across a fuel bed. *Actes de la conférence ESS 2001 conference*, pages 641–643. Marseille, France.

[Notton et al., 2002] Notton, G., Muselli, M., Cristofari, C., Poggi, P., et Louche, A. (2002). Intégration de systèmes hybrides à sources renouvelables d'énergie pour la production d'écentralisée d'électricité en sites isolés. *Actes du Congrès International Environnement et Identité en Méditerranée*, 1(1) :243–251.

[Notton et al., 2000] Notton, G., Muselli, M., Poggi, P., et Louche, A. (2000). Optimization of stand-alone hybrid pv/engine generator/battery system for small energy load, a case study in corsica. *PV Hybrid Power Systems 2000 Conference*.

[Nutaro, 2003] Nutaro, J. (2003). Page du projet adevs. http ://freshmeat.net/projects/adevs/.

[Orën, 1991] Orën, T. I. (1991). Dynamic templates and semantic rules for advisors and certifiers. *Knownledge Based Simulation : Methodology and application.*, pages 53–76. Springer Verlag.

[Oussalah et al., 1995] Oussalah, A., Magnan, M., et Talens, G. (1995). Emost : Modélisation par objets de systèmes techniques. *Génie Logiciel*, 1(37) :31–44.

[Rahman et al., 2001] Rahman, J., Cuddy, S., et Watson, F. (2001). Tarsier and Icms : Two approaches to framework development. *Actes du Congrès International Congress on Modelling and Simulation (MODSIM 2001)*, 1 :1625–1630.

[Risebro et Holden, 2002] Risebro, N. et Holden, H. (2002). *Front Tracking for Hyperbolic Conservation Laws*. Academic Press.

[Risebro et Tveito, 1992] Risebro, N. et Tveito, A. (1992). A front tracking method for conservation laws in one dimension. *J. Comp. Phys.*, 101 :130–139.

[Rizzoli et al., 1998] Rizzoli, A., Davis, J., et Abel, D. (1998). A model management system for model integration and re-use. *Decision Support Systems*, 4(2) :127–144.

[Sarjoughian et Zeigler, 1998] Sarjoughian, H. et Zeigler, B. (1998). Devsjava : Basis for a DEVS-based collaborative ms environment. Dans *Actes de la conférence 1998 SCS International Conference on Web-Based Modeling and Simulation*, volume 5, pages 29–36. San Diego, CA.

[Sarjoughian et Zeigler, 1999] Sarjoughian, H. et Zeigler, B. (1999). Collaborative modeling : The missing piece of distributed simulation. *Actes de la conférence SPIE, Enabling Technology for Simulation Science III*, 3696 :126–135. Orlando, FL.

[Schattenberg et Uhrmacher, 2001] Schattenberg, B. et Uhrmacher, A. (2001). Planning agents in James. *Actes de la conférence IEEE transacions on modeling and computer simulation*, 89(2) :158–173.

[SISO, 2003] SISO (2003). Site internet de l'organisation des standards d'interopérabilité en simulation, (simulation interoperability standards organization). http ://www.sisostd.org.

[Stella, 2003] Stella (2003). High performance systems, STELLA. http ://www.hps-inc.com/.

[Stimpfl-Abele, 1995] Stimpfl-Abele, G. (1995). Validation of input data for trained neural-nets. *Computer Physics Communications*, 85(2) :176–188.

[SWARM, 2002] SWARM (2002). Swarm development group. http ://www.swarm.org/.

[Vangheluwe, 2000] Vangheluwe, H. (2000). DEVS as a common denominatorfor multi-formalism hybrid systems modelling. *Actes de la conférence IEEE International Symposium on Computer-Aided Control System Design*, 1(1) :129–134.

[Vangheluwe et al., 2002] Vangheluwe, H., Lara, J., et Mosterman, P. (2002). An introduction to multi-paradigm modelling and simulation. Dans *Actes de la conférence AIS 2002 Conference*, volume 1, pages 9–20. Lisboa, Portugal.

[Wainer, 2003] Wainer, G. (2003). Liste des outils basés sur le formalisme DEVS. http ://www.sce.carleton.ca/faculty/wainer/standard/tools.htm.

[Wainer et Giambiasi, 2001] Wainer, G. et Giambiasi, N. (2001). Application of the Cell-DEVS paradigm for cell spaces modelling and simulation. *Simulation*.

[Web3D, 2003] Web3D (2003). Web3d consortium. http ://www.web3d.org/x3d.html.

[Woodbury et al., 2002] Woodbury, P., Beloin, R., Swaney, D., Gollands, B., et Weinstein, D. (2002). Using the ECLPSS software environment to build a spatially explicit component-based model of ozone effects on forest ecosystems. *Ecological modelling*, 150(3) :211–238.

[Wooldridge, 2002] Wooldridge, M. (2002). *Introduction to MultiAgent Systems*. John Wiley and Sons.

[Wymore, 1977] Wymore, A. (1977). *A Mathematical Theory of Systems Engineering - The elements*. Krieger Publishing.

[Zeigler, 1984] Zeigler, B. (1984). *Theory of Modeling and Simulation*. Krieger Publishing Company. 2nd Edition, ASIN : 0471981524.

[Zeigler, 1998] Zeigler, B. (1998). DEVS theory of quantization. Rapport Technique N6133997K-0007 ECE Dept., U Arizona, DARPA.

[Zeigler, 2000] Zeigler, B. (2000). *Theory of Modeling and Simulation*. Academic Press. 2nd Edition, ISBN : 0127784551.

[Zeigler et al., 1999] Zeigler, B., Hall, S., et Sarjoughian, H. (1999). Exploiting HLA and DEVS to promote interoperability and reuse in lockheed's corporate environment. *SIMULATION, Special Issue on The High Level Architecture*, 74(4) :288–295.

[Zeigler et al., 1996] Zeigler, B., Moon, Y., Lopez, V., et Kim, J. (1996). DEVS approximation of infiltration using genetic algoritm optimization of a fuzzy system. *Mathematical and Computer Modelling*, 23(11/12) :215–228.

Liste des figures

Liste des algorithmes

Annexe : Réalisations logicielles

Cette annexe présente les principales réalisations logicielles effectuées durant la thèse, ces logiciels sont disponibles sur support CD-ROM (attaché au manuscrit) ou bien directement sur la page http ://www.batti.org/these/. Le CD-ROM se parcoure grâce à un interpréteur HTML (Netscape, Mozilla, Konkeror, Internet explorer, ...) en ouvrant le fichier index.html situé à la racine du CD-ROM.

Plusieurs logiciels sont disponibles à partir d'un menu disposé dans une colonne à gauche de la page. Il est composé de six sections :

- JDEVS, Application, contenant aussi les quelques modèles présentés (Photovoltaïque, mouches de fruits),
- modèles basés sur les automates cellulaires,
- modèles basés sur les automates cellulaires mais en multicomposants,
- modèle très simple comparatif entre la technique cellulaire et Vector-DEVS,
- applications d'extraction de connaissance développées pour le Ministère de l'Economie des Finances et de l'Industrie dans le cadre d'un contrat de recherche,
- applications de modélisation de série temporelle par réseau de neurones (perceptron multicouche).

Ces logiciels sont tous expérimentaux mais fonctionnels. Ils constituent les prototypes qui nous

ont permis de définir le cadriciel et les techniques de modélisation et de simulation développés dans le cadre de la thèse.

Un soin certain a été apporté à la réalisation de ces logiciels, cependant, ils ne sont pas exempts de quelques bogues, et le code source est peu commenté.

Nous avons choisi de publier les prototypes plus stables, le lecteur pourra remarquer que ces versions ne sont pas rigoureusement conformes au cadriciel final présenté dans la thèse, qui est implémenté dans une version encore non publiable.

Ces logiciels utilisent le language Java et l'extension Java3D, disposer de Java Runtime Environment permets de lancer des exécutables Java (.jar) comme n'importe quel autre exécutable quelque soit la plate-forme cible.

Ce document a été rédigé en utilisant LaTeX2e.

http://www.tug.org/teTeX/

Une architecture logicielle pour la multi-modélisation et la simulation à évènements discrets de systèmes naturels complexes

Résumé :

La modélisation informatique est un outil essentiel à l'étude de systèmes naturels complexes. Les modèles sont souvent construits par des spécialistes et la diversité des techniques de conception utilisées, les rendent difficilement compatibles. Or, pour pouvoir étudier un système dans sa globalité, il est nécessaire d'agréger ces modèles en un multi-modèles. Il existe donc un besoin flagrant d'approche logicielle, en matière de conception de multi-modèles, ainsi qu'un besoin connexe de techniques adaptées à l'étude de systèmes naturels.

Notre travail apporte une contribution à la définition d'une telle approche. Nous nous appuyons sur l'analyse orientée objet pour déterminer les composés nécessaires d'une architecture logicielle, ou cadriciel, pouvant atteindre ce but. D'un point de vue formel nous utilisons le formalisme DEVS (Discrete EVent System specification) comme base unificatrice assurant la compatibilité des modèles. L'originalité de l'approche consiste à séparer toute technique de modélisation spécifique des composés de base. Chaque modèle créé partage ainsi un ensemble minimal de propriétés quelle que soit la technique utilisée. Trois techniques spécifiques d'étude de systèmes naturels ont été intégrées. La première, Feedback-DEVS s'intéresse aux modèles auto-apprenants ; la seconde, par automates cellulaires permet d'étudier la dynamique de systèmes spatialisés ; la troisième, Vector-DEVS, est une méthode originale d'étude de phénomènes à propagation d'interface sur cartes vectorielles. Pour chaque technique nous avons détaillé la spécification formelle et l'analyse objet. L'approche a été ensuite mise en pratique par l'implémentation du cadriciel, JDEVS et validée par trois expérimentations.

A software architecture for multi-modeling and discrete event simulation of natural complex systems

Abstract :

Computer models are essential as tool to study the behavior of complex natural systems. Specific models of parts of the system are often build by domain specialists and the diversity of techniques that they use makes the resulted models hardly compatibles. Nevertheless, to be able to study the entireness of a system, it is necessary to gather all specific models in a multi-model. Therefore it exists a strong need for a multi-modeling software approach, and for modeling techniques adapted to study natural systems.

Our work propose a contribution for such approach. We have used object oriented analysis to determine the appropriate components of a software architecture, or framework, that reaches this goal. From a formal perspective, we are using DEVS (Discrete EVent System specification) formalism as a unifying formalism to ensure models compatibility. The originality of our framework is the explicit separation of the technique specific modules and the core modules. This separation ensure that whatever the technique used, each model will share a minimum set of properties that is sufficient to enable coupling. Three specific techniques as been added to the framework. The first one, Feedback-DEVS is focused on self-learning models ; the second, cellular automata, enable to study spatially explicit models ; the third, Vector-DEVS, is an original technique that permits to study the evolution of the interface of a phenomenon on vector maps. For each of the techniques we have provided the formal specification as well as the object oriented analysis of the resulted modules. Finally, we have implemented a software based on this framework, JDEVS and performed three experimentation to validate our approach.

Discipline, Spécialité : Sciences pour l'Environnement, Informatique

Mots clés : multi-modélisation, modélisation, simulation, logiciel, DEVS, évènements discrets, propagation d'interfaces, automates cellulaires, Java, orienté objet, XML.
Keywords : multi-modeling, modeling, simulation, software, DEVS, object oriented, discrete events, front tracking, cellular automata, XML.

SPE (Systèmes Physiques pour l'Environnement), UMR CNRS 6134
Quartier Grossetti, BP 52, 20250 Corté
http://spe.univ-corse.fr